Large-Scale Parallelised Boundary Element Method Electrostatics for Biomolecular Simulation

David R. Fallaize

UCL

A thesis submitted for the degree of

Doctor of Philosophy

February 2012

Declaration

I, David R. Fallaize, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

David Fallaize
February 2012

Abstract

Large-scale biomolecular simulations require a model of particle interactions capable of incorporating the behaviour of large numbers of particles over relatively long timescales. If water is modelled as a continuous medium then the most important intermolecular forces between biomolecules can be modelled as long-range electrostatics governed by the Poisson-Boltzmann Equation (PBE).

We present a linearised PBE solver called the "Boundary Element Electrostatics Program" (BEEP). BEEP is based on the Boundary Element Method (BEM), in combination with a recently developed O(N) Fast Multipole Method (FMM) algorithm which approximates the far-field integrals within the BEM, yielding a method which scales linearly with the number of particles. BEEP improves on existing methods by parallelising the underlying algorithms for use on modern cluster architectures, as well as taking advantage of recent progress in the field of GPGPU (General Purpose GPU) Programming, to exploit the highly parallel nature of graphics cards.

We found the stability and numerical accuracy of the BEM/FMM method to be highly dependent on the choice of surface representation and integration method. For real proteins we demonstrate the critical level of surface detail required to produce converged electrostatic solvation energies, and introduce a curved surface representation based on Point-Normal G1-continuous triangles which we find generally improves numerical stability compared to a simpler surface constructed from planar triangles. Despite our improvements upon existing BEM methods, we find that it is not possible to directly integrate BEM surface solutions to obtain intermolecular electrostatic forces. It is, however, practicable to use the total electrostatic solvation energy calculated by BEEP to drive a Monte-Carlo simulation.

Contents

List of Figures

List of Tables

Acknowledgements

I could not have completed this PhD without the help and support of various people. Firstly I would like to thank my supervisors, Dr. Mark A. Williams and Prof. John E. Ladbury, who provided intellectual support throughout the project, and were always available with useful advice. I would particularly like to thank Mark who, as my principal supervisor, gave me what felt like perfect freedom to pursue the project as I chose, whilst simultaneously providing the necessary guidance to prevent me from becoming hopelessly lost. Over the last four years I have enjoyed the benefit of Mark's considerable knowledge and experience and I am extremely grateful not just for his insightful comments, but also for his continuous encouragement. I would also like to thank Paul Driscoll who acted as my thesis committee chairman, as well as the past and present members of the Williams and Ladbury groups, who helped fill the gaps in my biological knowledge (often during coffee-breaks).

Secondly I would like to thank my family and friends who have encouraged my efforts over the last four years: my parents Robert and Cathy who have given me love and support throughout my life; my siblings Andrew, Claire and Mike who listened to me patiently, even when I was being boring or just complaining; Jonathan Kahn and Rhiannan Walton who kept me fed, watered and supplied with coffee on countless occasions.

Finally I would like to thank my wife Emma who has supported me, with seemingly infinite patience, throughout this project. This work is dedicated to Emma, with my love.

David Fallaize

February 2012

Chapter 1

Introduction

All life is fundamentally based upon the interactions between biological molecules (e.g. proteins; nucleic acids), which in human cells coexist in a complicated and crowded mixture. At the atomic length-scale and picosecond timescale, protein structures flex and explore their local energy landscape, constrained largely by short range covalent and hydrogen bond interactions. On a nanosecond timescale, proteins gradually diffuse through the cellular medium and encounter other proteins. On a microsecond to millisecond timescale, complexes form between different proteins. Over time, these complexes will further diffuse, interact, break apart and re-form in new combinations. At a microscopic level, it is the collective behaviour of many copies of the proteins and complexes which form an emergent system; for example, an intracellular signalling pathway activated in response to a cell-surface receptor detecting some external chemical signal may lead to a whole-cell level change in behaviour (e.g. cell division).

Isolated proteins can be studied in detail using X-ray crystallography and NMR to elucidate the 3D atomic structure. The emergent behaviour of proteins at a cellular level can be studied with microscopic techniques, on a millisecond timescale upwards (e.g. through fluorescent tagging and confocal microscopy). But there is a large gap in our understanding: how do proteins behave at the time- and length- scale of protein complex formation, in the crowded environment found within a cell? This question lies at a scale between those of the experimental approaches available to us, yet is crucial to understanding biological processes.

Computer-based experiments can fill this gap. Computer simulations are *models* of reality (with a distinct meaning from *replica* of reality), and intended to be tools which aid understanding, and suggest avenues for further experiment [1]. Nonetheless there is often an apparent scepticism surrounding the use of computational models in biology. A recent

review by van der Kamp *et al.*[2] opens with the question "Can I trust modelling?" and discusses the recent progress in computer modelling techniques (with emphasis on Molecular Dynamics), before pointing out the notable successes such methods have had in biomedical research: for example virtual screening of compounds for drug discovery [3]; protein-ligand docking studies [4]; the development of novel proteins for industrial use e.g. as catalysts [5]. A healthy mistrust of computer models is perfectly reasonable (after all they are ultimately based upon experimental data), however, there is no denying their utility. Perhaps an ideal computer simulation would reproduce reality *in-silico* and therefore always match experiments perfectly[1]; but the model required would necessarily contain all of the complexity of reality so, besides being impractical, the model would be no easier to understand than reality itself.

The original aim of this project was to enable a level of simulation or modelling on the scale of thousands of macromolecules within a subcellular compartment on timescales of up to seconds: this is orders of magnitude beyond the capabilities of current particle simulations such as Molecular Dynamics. In order to achieve this, a model of "reality" is required which takes into account the most important interactions between macromolecules whilst still remaining computationally feasible. This thesis concerns itself particularly with the representation of electrostatic interactions between macromolecules, within the larger context of a large-scale simulation.

We begin by discussing the important features of macromolecular space at the scale of simulation in which we are interested, followed by a short review of the established methods for biomolecular simulation. We suggest that the dominant consideration is the long-range electrostatic interaction between macromolecules, and summarise the state-of-the-art in the area of electrostatics for biomolecules.

1.1 Macromolecular Interactions

1.1.1 Intermolecular Forces

The dominant intermolecular forces acting on macromolecules in cytosol are entirely electrostatic in origin.

At short-range there is the attractive component of the van der Waals force which arises from so-called "flickering dipoles" which are set up between the electron distributions of nearby atoms: small fluctuations in electron distribution induce corresponding fluctuations in the electron distribution of neighbouring atoms, leading to an instantaneous asymmetry

[1] assuming that the results of the experiment also match reality with the same accuracy

in the charge distribution; a dipole. The dipoles will attract one another, leading to a small attractive force between the atoms/molecules. The attractive effect is very small and the atoms must be in close proximity for the effect to be significant. At still closer proximity, at the point where electronic orbitals of atoms of neighbouring molecules start to overlap, this attraction is overwhelmed by strong repulsion. The Lennard-Jones "6-12" potential [6] models this effect as a function of distance r between particles, with the functional form given in Equation 1.1.

$$U_{vdW}(r) = 4\alpha \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{1.1}$$

The 6th order term models the attractive component, whilst the 12th order term models the extreme repulsion at very short range; the equilibrium distance where the attractive/repulsive components are balanced is controlled by the term σ, and the depth of the energy well is controlled by α.[2]

A molecule (or atom) can hold a net charge through ionisation, which will interact by long-range electrostatics with all other molecules in the vicinity which feature a net charge, dipole, or higher order multipole distribution. More subtly, the bonding between atoms of differing electronegativity can lead to asymmetries in the electron distributions around atoms: one atom in the bonded pair acquires a small "partial charge" from the bond which is counterbalanced by the other atom. This charge asymmetry gives the bonded pair a "permanent dipole".

The effect of electronic asymmetry, taken over the entire molecule, can result in a complicated arrangement of dipoles and higher order multipoles distributed through the molecule. The permanent dipoles (and higher multipoles) produce long-range electrostatic interactions between molecules, even though the molecules themselves may have zero net charge.

1.1.1.1 Hydrogen bonds

In addition to charge-charge and charge-dipole interactions between molecules, there is another particularly strong form of non-bonded electrostatic interaction which can occur between a hydrogen atom on one molecule and a relatively electronegative atom (such as fluorine, oxygen or nitrogen) on an opposing molecule: this is called the hydrogen bond [7]. In order for this effect to take place, the hydrogen atom on the first molecule (termed the "donor") must itself be covalently bonded to an electronegative atom which results in an

[2] It is worth noting that the choice of a 12th order term for repulsion is mostly for simple implementation of the potential function, since the r^{-12} component is just the r^{-6} component squared. A different choice of exponent, such as r^{-10} or r^{-14} would be no less reasonable, but would require more effort to compute.

unusually large permanent partial positive charge on the hydrogen atom. The electronegative atom on the second molecule (termed the "acceptor") can then form a very strong bond with the partial positive charge of the hydrogen atom: in fact the lone-pair electrons of the acceptor become de-localized and partially associated with the hydrogen atom of the first molecule, resulting in an electronic configuration which starts to look a little like a covalent bond (though the strength of the interaction is somewhat less than that of a covalent bond).

The strength of a hydrogen bond varies according to the type of electronegative atoms involved, which controls the degree of electronic asymmetry on the donor hydrogen and the extent to which the electronic distribution on the acceptor atom can be reassigned to the hydrogen bond. In addition the hydrogen bond is highly directional, with maximum bond strength occurring when the three atoms are aligned linearly (180^{o} bond angle). Consequently although the hydrogen bond is energetically favourable, it can effectively impose some constraints to the orientations of atoms involved in the bond. The strength of the hydrogen bond includes both the "conventional" charge-charge interaction of the partial charges as well as the quantum-mechanical component of the electronic de-localization.

1.1.1.2 The strength of salt bridges: electrostatics within (and between) proteins

The exterior surfaces of proteins (as well as protein secondary structure itself) is controlled by the network of hydrogen bond interactions. Hydrogen bonds can also be formed between certain protein residues, such as between the carboxylate group of glutamic acid or aspartic acid and the ammonium group of lysine, arginine, tyrosine, histidine or serine. These are called "salt-bridges" and can take place between residues on the same protein, or across a protein-protein interface.

It is possible to estimate the strength of a salt-bridge by experiment or through computation. In both cases it is necessary to locate a salt-bridge on a molecule, and then carry out mutagenesis on the amino-acid residues (either in reality or virtually), such that one has data for the "wild-type" and for single mutants where one residue in the salt-bridge pair is substituted for a non-hydrogen bonding residue side-chain, and for the double mutant where both residues are mutated.

For theoretical calculations it is necessary to solve the electrostatics of the protein in each case (e.g. by solving the Poisson equation: see Section 1.12). The electrostatic component of a salt-bridge can be deduced by the differences in solvation energies between the mutants and wild-type configurations, as demonstrated by Hendsch and Tidor [8].

To calculate the strength of salt-bridges by experiment is slightly more complicated. The general methodology is to run a titration experiment in an NMR machine which measures

the chemical shift for protons in the vicinity of the salt bridge. As the pH of the solution is increased from a low value, the donor protons in the salt-bridges (i.e. a hydrogen atom in the ammonium ion) will gradually transfer to the bulk solution, until all salt bridges on the proteins in solution have been broken. As this transition occurs, the NMR signal for protons attached to the carboxylate groups of the salt bridge will show a change in chemical shift. By measuring the pH at which 50% of salt-bridges have been broken (i.e. the midpoint of the chemical shift curve with respect to pH), it is possible to find the pK_a for the donor proton of the salt bridge.

Repeating the experiment for the mutants gives the change in pK_a for each mutant. The pK_a is related to the Gibbs free energy (ΔG) by the gas constant (R) and absolute temperature T, and the equilibrium dissociation rate constant K_{eq} such that: $\Delta G = -RTln(K_{eq})$, and $pK_a = -log_{10}(K_{eq})$. Thus changes in pK_a can be related directly to a change in free energy associated with the removal of the salt-bridge (with the stability of the salt bridge being the change in energy for the double mutant relative to the wild-type, minus the free energies for each of the single mutants relative to the wild-type).

In looking at results for salt-bridge energies from either calculation or pK_a shifts, it is extremely important to know what the reference state is: for an example salt-bridge in T4 lysozyme (histidine-aspartic acid) Hendsch and Tidor conclude that the salt-bridge is *de*stabalising on the order of around 1 kJ/mol, whilst NMR experiments by Anderson *et al.* [9] tend to suggest that the same salt-bridge is stabalizing, with a favourable energy of around between 0.72 and 1.20 kJ/mol. The discrepency relates to what the salt-bridges are being compared to: Hendsch and Tidor argue that the correct comparative state is a pair of hydrophobic residues of the same size as those in the salt-bridge, whilst the practical experiment compares the salt bridge to a pair of arginine residues. In the former case, it is energetically unfavourable to insert polar groups in a partially buried surface, even if they can form a salt-bridge. However, once a pair of polar groups are present, as in the latter case, it is energetically more favourable for them to form a salt-bridge than not.

1.1.2 The importance of water

1.1.2.1 Water: a polar molecule

Water is a polar molecule with a strong permanent dipole, capable of forming hydrogen bonds as either acceptor (via the central oxygen atom) or donor (via the two hydrogen atoms). Since water molecules in bulk solution readily form a network of hydrogen bonds, it is, by comparison, energetically unfavourable for water to be located anywhere else which does not feature the same degree of hydrogen bonding potential. Thus whilst polar surfaces

of biological macromolecules, e.g. proteins, can form reasonably energetically favourable water contacts, apolar surfaces will tend to aggregate together, since it is energetically more favourable for water molecules to be in bulk solution than adjacent to an apolar surface. This is the "hydrophobic effect".

The hydrophobic effect is also important for the interaction between proteins in solution at short range. In order for two solvated (polar) surfaces to come together (such as in the case of a small ligand docking into the enzymatic cavity of a larger protein, or in non-specific protein-protein interactions) the water molecules associated with those surfaces must be removed. The release of water molecules back into the solution may be slightly energetically favourable, since the water will readily replace any hydrogen bonds previously made with the protein with equivalent hydrogen bonds in the bulk solvent, and not all water molecules will have made hydrogen bonds to the protein surface. However the restriction of the surface atoms as they come together will lead to an entropic penalty, which must be compensated by an equivalent gain in energy elsewhere. At best, the two surfaces will form (between them) the same number of hydrogen bonds as when solvated with water molecules, so the enthalpic penalty may be negligible. More likely, the total number of hydrogen bonds will decrease, leading to both enthalpic and entropic penalties, which must somehow be compensated by the total energy of the combined surfaces, e.g. through more complete burial of apolar surface or more favourable electrostatics between the combined surfaces.

Even if the overall transition from solvated surfaces to desolvated complex is energetically favourable, the height of the energy barrier presented by desolvation[3] will have a significant effect on the kinetics of the process; if there is insufficient thermal energy to drive the process then the complex may not form at all during the lifetime of the protein components.

From a simulation point of view, the energy changes associated with the solvation of macromolecules, and the effects of desolvation at short-range, in principle could be treated as an "average" intermolecular force. However it is difficult to know how this effect could be correctly parameterised, since the behaviour of water in these conditions is expected to be different from that of bulk water; consequently obtaining the physical data with which to parameterize the relevant forces would be difficult.

Apart from the short-range effects of water, which arise primarily from hydrogen bonding, water has a long range electrostatic effect on the interaction between macromolecules: water at room temperature has a dielectric constant of about 78, so intermolecular electrostatic forces between charged macromolecules are much reduced by the presence of water.

[3]Regardless of the relative degree of hydrogen bonding of individual water molecules at the protein surface vs. those within bulk water, the presence of water molecules at the surface of the protein constitute a polarised charge distribution interacting electrostatically with the electric field of the protein: the "solvation energy" represented by the interaction between protein and polarised solvent is significantly large, as we will see throughout this thesis.

1.1.2.2 Water: driving diffusion

The other major effect of water on the behaviour of macromolecules in solution is as the driver of diffusion. Treated on a long enough timescale, and a large enough length-scale, the random collisions between individual water molecules (small) and macromolecular solutes (large) gives rise to Brownian motion of the macromolecules.

The hydrodynamic properties of a macromolecule in solution can be modelled by its diffusion tensor[4], and for isolated bodies this can be estimated by a variety of means: bead models [10], shell models [11] or boundary element methods [12, 13].

The motion of multiple macromolecules in solution can become very complicated, since the water molecules which drive the motion are not totally independent, and correlations between particles is transmitted through the medium by hydrodynamic interaction. Strictly the diffusion tensor of the whole system should be found, which will vary according to the relative position and orientation of all macromolecules; the components of the grand diffusion tensor will contain the individual diffusion tensors of each macromolecule, with additional off-diagonal tensors describing the pairwise hydrodynamic interactions between them.

In practise, the treatment of hydrodynamic interactions depends on the simulation methodology used. Explicit water methods such as MD should incorporate the hydrodynamic effect automatically by virtue of treating all water-water interactions. In implicit solvent simulations of proteins in solution, the hydrodynamic interaction is generally considered to be of less importance to the motion of macromolecules than the electrostatic interactions [14, 15], therefore in such simulations (such as Brownian Dynamics) particles are usually each assigned a constant diffusion behaviour (either a full tensor, or just a single diffusion coefficient to model isotropic diffusive behaviour). In such methods the hydrodynamic interaction is neglected, as it has been computationally too expensive to rigorously calculate hydrodynamic interactions using the diffusion tensor.

Although there is some evidence to suggest that hydrodynamic interactions are relatively unimportant in general compared to electrostatics, there is also some evidence to the contrary for specific cases such as small molecule protein-ligand binding, in which the hydrodynamic interaction between the molecules can have a substantially greater effect on association rate than electrostatics, through "steering" of the ligand to its binding site [16].

In particular, it seems likely that in a crowded and concentrated simulation, the hydrodynamic interaction which is considered small for two close entities at otherwise near-infinite dilution, may become significant.

[4] a rank-2 tensor; so can be thought of as a conventional matrix

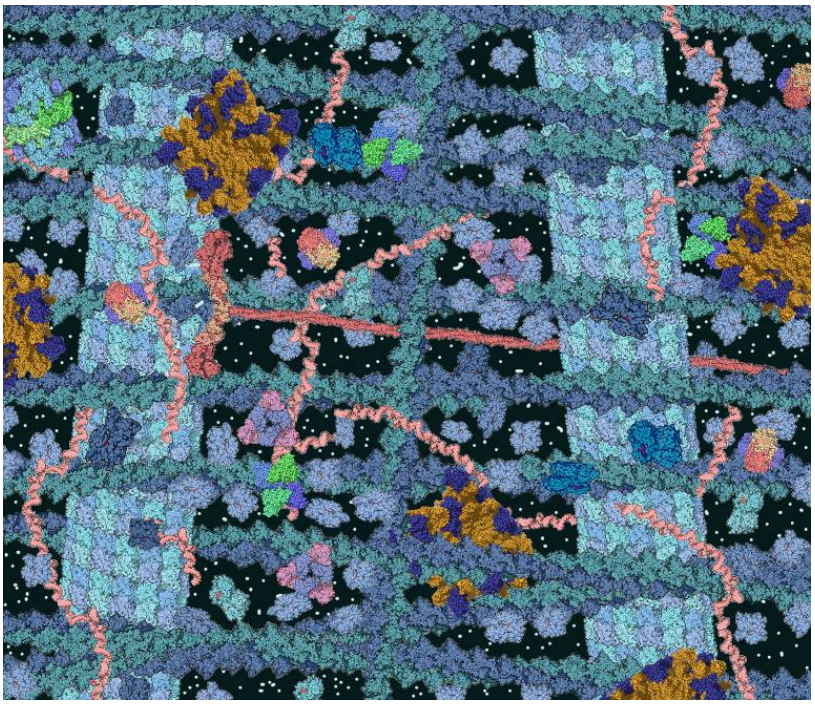


Figure 1.1: Can we simulate this? Artist's impression of the interior of a cell, illustrating diverse range of biological structures (each tens of thousands of atoms in size): microtubules (blue), actin (dark blue), ribosomes (yellow/purple), soluble proteins (light blue), RNA (pink). [Public domain image by Tim Vickers, based on similar illustrations by David Goodsell [17]].

1.1.3 Macromolecular Crowding

Cells are between 8-40% protein by mass, which means a significant volume of space is occluded by the macromolecules, leading to restricted diffusion and an enhancement of effective local concentration [18]. The idea that cells are very crowded environments has been well appreciated for several decades, however almost all biochemical characterisations of biomolecules necessarily takes place at high dilution compared to the effective concentration of macromolecules found in the native environment. Simulation methods have in the past been restricted to a relatively small number of particles by computational constraints, so often the computational approach is in someways as limited as the experimental.

The idea that simulations and experimental methods need to take the effect of cellular crowding seriously seems to be gradually gaining support [19]. In order to explicitly simulate crowding effects, it is necessary first for a simulation to be able to simulate large numbers of interacting macromolecules. An illustration of what this entails is given in Figure 1.1, which depicts the diverse range of macromolecular structures in a real biological system, which we would like to be able to model computationally.

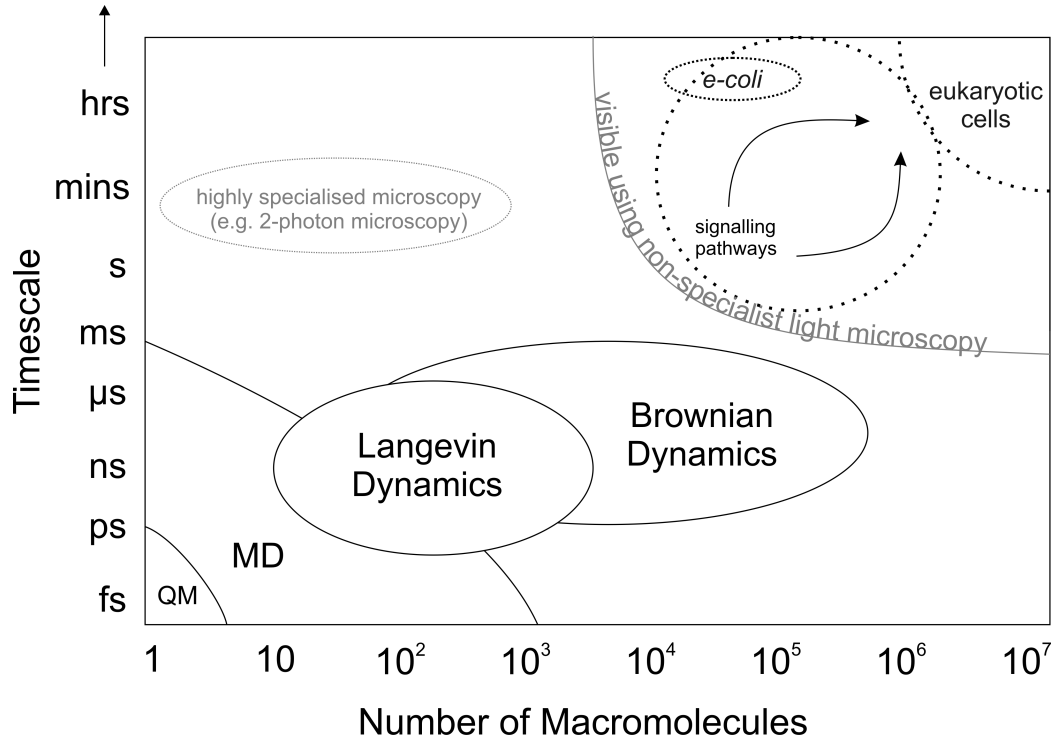


Figure 1.2: Time- and length- scales for biological simulations. The upper right region represents the approximate scales for directly observable biological processes. Current particle simulation methods are orders of magnitude short of this level.

1.2 Scales of simulation

There are many methods for computationally simulating biological processes. Simply speaking, these methods can be categorised according to the time- and length- scales on which they operate, as illustrated in Figure 1.2.

At the lower length and timescale the methods are particle-based and atomically detailed, for example Quantum Mechanics and Molecular Dynamics. As the scales are increased, detail is necessarily removed from the simulation method (e.g. Langevin Dynamics and Brownian Dynamics (BD) where individual water molecules are replaced by a continuum model. This level of simulations (on the µm length-scale, and ms timescale upwards) is not well covered in simulation-space, as the computational requirements have historically been too high to enter this region. This is unfortunate, since it is at this scale that cellular processes such as signalling pathways operate; in effect it is the scale at which "state" is encoded into a cell, through the instantaneous mix of macromolecules which give rise to cellular behaviour [20], and everything which cell biologists can observe are a consequence of processes occurring at this level.

At the highest level of simulations illustrated in Figure 1.2, the simulations cease being particle-based and the individual macromolecules are replaced by numerical approximations

of their aggregate behaviour; for instance, concentrations and rate constants within a system of ordinary differential equations which describe the evolution of these quantities over time. Examples of this are models of "simple" calcium concentration dynamics in neuronal cells [21] and the more ambitious modelling of entire regulatory networks [22].

There are various methods available for modelling biological phenomena at this level as stochastic processes. Green's Function Reaction Dynamics, described by Zon and Wolde [23] uses an analytical solution to the Smoluchowski equation (the governing equation of stochastic particle dynamics) to generate probable time intervals between reaction events in a dilute solution of reactants. Alternatively the novel method described by Tournier *et al.* [24] use conditional probabilities of interaction, such as those generated by a reaction-diffusion Brownian Dynamics simulation, to timestep a larger scale simulation.

The recent trend in Systems Biology is to then connect this level of model into a multi-scale hierarchical model of, e.g. a human in the Virtual Physiological Human project [25]. This relies on connecting models of lower-level systems together in order to obtain a model of the whole, using some common *lingua franca* to represent and document the parameters being passed between separately developed sub-models, e.g. using CellML [26]. Whilst the output of this has enormous potential value to for treatment of human illness, the validity of the output of the model will ultimately depend entirely on the quality of the parameterisations for the sub-models on which it depends.

Generally those sub-models are of the high-level ODE type described above: numerical constants replacing the behaviour of large numbers of biomolecules. These high level simulations all rely on parameters intended to describe the behaviour of a very large number of macromolecules in solution, e.g. concentrations, rate constants, affinities. Sometimes these quantities can be measured experimentally e.g. the change in enthalpy for the complex formation between a protein and ligand can be measured using isothermal titration calorimetry (ITC), the results of which can be used to derive thermodynamic quantities describing the overall interaction; useful parameters such as equilibrium dissociation constants can be derived from such experiments. The weakness of such parameterisation is that the measurements do not necessarily correspond to the true biological environment within a cell, e.g. ITC is carried out with physiologically unrealistic quantities/concentrations, in isolation from other cellular crowding agents. Where models are parameterized from "direct" observation of a cell (e.g. rates of change of concentration) there is more confidence that the parameters really do capture the relevant behaviour of a "real" cell; however there is no way to link this parameter to the underlying molecular process which brings it about.

It is here that the role of large-scale simulations becomes clear. Apart from the general utility offered by large scale simulations (better understanding of cellular processes which

operate on that scale), computer simulations lying in the empty space of Figure 1.2 have the potential to augment the experimentally measured parameters currently used in higher level simulations, and connect the structural information of individual proteins to the high-level simulations of an entire human.

1.3 Particle Simulation Methodologies

1.3.1 Quantum Mechanics

At the very finest level of detail are quantum mechanical (QM) approaches which describe the electronic wavefunction of each atom in the molecule. Methods such as density functional theory (DFT) (see e.g. Leach[27] for a summary) are commonly used to enumerate the energy states of small collections of chemically bonded atoms. Such *ab-initio* computations give accurate results and explicitly incorporate the precise electrostatic nature of the system (rather than dealing with their net effect via atom-centred partial charges and dielectric constants) but at very high computational cost which prohibits using such methods for very large macromolecules or for long timescale simulations.

QM models can be incorporated into a higher level "molecular mechanics" methodology such as MD (yielding QM/MM, pioneered by Warshel and Levitt [28]) in order to accurately simulate some feature of the system (such as the active site of an enzyme) which the MD parameterisation may not be able to adequately capture. The QM part of the algorithm evaluates quantum mechanical properties of the atoms within a certain region, which are then added to classical descriptions of the energy terms for the remainder. The intention is to achieve a method which combines speed of operation with improved accuracy for the parts of the molecule which are of primary interest or importance.

1.3.2 Molecular Dynamics

Molecular dynamics (MD) is a widely used and well established technique for simulating atoms and molecules. Some excellent reviews of MD in general have been written, for example those by Adcock and McCammon [29], and van der Kamp [2].

Essentially an MD solver (such as AMBER [30], NAMD [31], GROMACS [32], CHARMM [33], TINKER [34]) evaluates Newton's second law (Equation 1.2) for all atoms in a simulation, given force constraints on the various atoms in the form of stretchable and rotatable bonds, and electrostatic repulsion/attraction.

$$F = ma = m\ddot{s}(t) = -\nabla U_{total}(s(t)) \tag{1.2}$$

where $s(t)$ is the position of atoms with mass m at some time t, whose acceleration a (which is the second derivative of position with respect to time, $\ddot{s}(t)$) . The force is equal to the negative gradient of the total potential field at position $s(t)$, at time t, denoted $-\nabla U_{total}(s(t))$. The total potential U_{total} represents the potential due to all interactions, both bonded and non-bonded, and includes the effect of solvent atoms.

The forces on the atoms lead to accelerations, which lead to changes in the velocities of each atom, from which new positions after a short timestep, $s(t + \delta t)$, can be calculated. There are various energy terms to be considered. The velocities of the atoms lead to a quantity of kinetic energy being associated with each molecule. The changes in relative positions of all of the charged atoms leads to a change in the overall electrostatic potential of the system with time.

During the simulation the MD solver tries to select configurations of atoms which avoid the total energy terms[5] rising (which, physically speaking, would correspond with a spontaneous rise in the temperature of the system). The limiting size of an MD timestep is the highest vibrational frequency of an atomic bond in the system, which is somewhere in the femtosecond range. Thus many steps of the MD algorithm are required to reach a biologically relevant length of simulated time.

A key consideration in carrying out MD simulations is the choice of "force field", i.e. the particular parameterization and functional forms describing the various energy terms associated with the atomic bonds within molecules (e.g. bond lengths, bond angles, torsion angles, preferred dihedrals) and the non-bonded interactions (electrostatic potentials, the Lennard-Jones potential). Many distinct force fields (with numerous variants) are available; a recent review of the major parameter sets is given by Mackerell [35]. Although many force fields are available, they are in some senses equivalent in that most of the quantities they encode are common across MD implementation. In general, the parameterization of a force field is carried out with reference to some idealised gas phase QM calculations, combined with a degree of "parameter tweaking" until the final results of the MD simulation match some experimentally measurable quantity (e.g. solvation energies of amino acid sidechains). The force fields differ in the choice of data used for this parameterization; successive generations of force field are commonly compared to the physical data available and minor adjustments made to correct for shortcomings exposed by new physical comparisons (e.g. Showalter *et al.* [36] found that the Amber99sb force field gave improved agreement with NMR data for model peptides compared to Amber99, while Thompson e*t al.* [37] claim that the Amber99Φ

[5] i.e. the total kinetic energy implied by the velocities of atoms, the potential energy of atomic bonds, plus the potential energy arising from non-bonded electrostatic interactions and Lennard-Jones interactions.

force field fixes deficiencies in Amber99sb[6]).

A comparison of MD force fields applied to a fully solvated molecule of bovine pancreatic trypsin inhibitor (BPTI) is given by Kim *et al.* [38]: the conclusion is that whilst the behaviour of the non-polar parts of the protein are in general agreement between force fields, the structure of the water near to charged surface regions of the protein is highly dependent on the force field used for both protein and water.

A perhaps unfortunate feature of MD is the simplicity with which the user can select a force field, without having any knowledge whatsoever of the assumptions and caveats that apply to that specific parameter set. Therefore although it is comparatively easy to carry out an MD simulation (download software; manipulate an input file; run), it is less easy to justify the results if one does not know what assumptions were made.

Despite this "opacity" associated with the choice of force field, MD is considered by most to be the best method currently available for biomolecular simulations. That is, given the constraints of time and/or computational resources available to the average biologist, it seems to give (generally) sensible output, and people find it a useful tool. Apart from the biomedical uses mentioned previously [2], MD is very widely utilized, including such applications as refining X-ray crystal structures and energy minimising NMR structures [39]; investigating protein folding/unfolding and stability [40]; simulating ion channels and membrane proteins [41].

There are two major drawbacks with the technique: first is the short timestep mentioned earlier; second is in the representation of water in the simulations. In the early days of MD all simulations were carried out in a virtual vacuum [42], since there was no computational power to spare for water molecules. The field of continuum electrostatics (discussed later in this chapter) was in part stimulated by the need to incorporate some correction into early MD models to account for the missing water, through "implicit water" models. The increased computing power now available means MD is generally carried out with "explicit water" since the expectation is that this should give maximum accuracy.

"Coarse graining" of MD is a method by which some atoms (typically an amino acid of a protein) are grouped together into a lump of effective mass and charge instead of being independently simulated [43]. This reduces some of the degrees of freedom within the simulation, and if high frequency modes of vibration are reduced then the length of timestep can also be increased. The penalty is a presumed loss of accuracy since the coarse-grained groups no longer interact quite so "realistically" as they should.

[6]It is suggested that the parameters in the Amber99sb force field destabalize alpha-helices whereas Amber99Φ produces results for model helical peptides that are in closer agreement with experimental results.

Through advances in computational power (e.g. the advent of GPU computing, and very large scale distributed computing such as the folding@home project [44]) it has become possible to carry out what a few years ago would be considered extraordinarily large MD simulations, for example the simulations carried out by D.E. Shaw Research [45]. However MD still remains many orders of magnitude short of the time and length-scale we require for large-scale macromolecular simulation of cellular processes.

1.3.3 Langevin Dynamics

Langevin dynamics is an implicit-solvent model for particle dynamics with equation of motion given by Equation 1.3 (using the same notation as Equation 1.2):

$$F = M\ddot{s}(t) = -\nabla U_{ns}(s(t)) - \gamma m\dot{s}(t) + R(t) \tag{1.3}$$

where U_{ns} represents the potential (bonded and non-bonded) due to everything *except* solvent molecules; γ is damping constant incorporating solvent viscosity, giving a velocity-dependent friction, $-\gamma m\dot{s}(t)$, and $R(t)$ is a random term arising from the solvent molecules colliding with the particle. $R(t)$ describes a random walk process controlled by the diffusion tensor; the equations controlling $R(t)$ are the same as those for Brownian Dynamics which is discussed in more detail in the Section 1.3.4.

This type of simulation is in effect very similar to MD (and in fact most MD packages incorporate options for performing Langevin Dynamics): importantly the solvent which is usually explicitly represented in MD is replaced in Langevin Dynamics with the damping factor and the random solvent collision term. The absence of solvent interactions in the potential field U_{ns} means that simulations will run faster than MD since there are far fewer pairwise interactions to calculate, but the timestep is still limited by the requirement to accurately model all changes in velocity of all components (though the timestep for Langevin dynamics can be increased to e.g. 12 fs [46] without impairing the observed dynamics). Therefore this method is still not suitable for the purposes of very large scale simulations. Apart from a few specialised uses (e.g. sampling counter-ion distribution around a protein [47]) the use of Langevin Dynamics for the simulation of protein systems does not seem widespread compared to the more conventional approach of explicit water MD.

1.3.4 Brownian Dynamics

Brownian Dynamics (BD) is a generalisation of Langevin Dynamics in that the average acceleration is assumed to be zero ($\dot{\bar{V}}(t) = 0$ therefore $F = M.\dot{\bar{V}}(t) = 0$) leading to the

equation of motion for Brownian Dynamics (Equation 1.4), where the mean values of the three terms on the right hand side of equation 1.3 sum to zero.

$$0 = -\overline{\nabla U_{ns}(s(t))} - \gamma m\overline{\ddot{s}(t)} + \overline{R(t)} \tag{1.4}$$

From this, Ermak and McCammon [48] showed that the translational behaviour of a particle under Brownian Dynamics can be written as Equation 1.5 which describes how, given a configuration of particles at the start of a timestep, one can calculate a new configuration for a short timestep Δt.

$$s_i = s_i^0 + \sum_j \frac{\partial D_{ij}^0}{\partial s_j}\Delta t + \sum_j \frac{D_{ij}^0 F_j^0}{k_B T}\triangle t + R_i(\Delta t) \tag{1.5}$$

where s_i is value of one translational/rotational degree of freedom (of one particle) in the simulation; superscript 0's denote that those values are taken at the beginning of the timestep, which is of length Δt. D_{ij} is a component of the grand diffusion tensor which describes the diffusivity of the entire set of particles in the simulation; $k_B T$ is the Boltzmann constant multiplied by absolute temperature; F_j is the force (or torque) component for the j^{th} degree of freedom in the simulation, and summations over the index j denote summations over the entire set of degrees of freedom of the system (of which there will be $6p$ for a system of p particles, each having three spatial and three rotational degrees of freedom; internal vibrational degrees of freedom are ignored since the particles are assumed to be rigid). Finally the term $R_i(\Delta t)$ is a Gaussian random-walk term modelling the diffusion of the particle in the direction of the i degree of freedom, such that the covariance of the random-walk obeys the statistical properties of Equation 1.6 and Equation 1.7.

$$\langle R_i(\triangle t) R_j(\triangle t)\rangle = 2D_{ij}^0 \Delta t \tag{1.6}$$

$$\langle R_i(\triangle t)\rangle = 0 \tag{1.7}$$

The advantage of this algorithm over Langevin dynamics is that the timestep can now be much longer: we are not concerned with internal molecular details such as stretching of inter-atomic bonds; nor are we required to track and scale the velocity of every particle at each timestep. The water molecules are all implicitly represented so, in common with Langevin Dynamics, there is a reduction in the number of particles to simulate relative to MD.

In fact, there is now a lower limit to the length of timestep: it must be sufficiently long for the "over-damped" Langevin condition to apply – i.e. any net acceleration during

a timestep must have time to decay away during that timestep. Of course, this is not a burden since we want a long timestep. The upper limit to the timestep for BD comes from the mean collision rate of the particles: the timestep must be short enough that particles don't end up inside each other in one timestep. In addition the timestep needs to be small enough that the change in long-range intermolecular force on each particle is reasonably small over one step.

Equation 1.5 is commonly simplified by assuming there are no hydrodynamic interactions between particles: this removes the spatial derivative term of the diffusion tensor, and allows the diffusion tensor D for each molecule to be treated separately and independently.

Unlike MD and Langevin Dynamics, the BD algorithm does not in itself have any means to control the 'temperature' (energy) of the system, since there is no concept of velocity of the particles. The energy of the system is the electrostatic energy plus internal energies of the molecules (which, if the components are assumed to be rigid, will not change over the course of a simulation). Consequently a Monte-Carlo accept/reject criteria is required for each timestep to maintain stability of the simulation, leading to so-called "Smart" Monte-Carlo Brownian Dynamics [49].

Brownian Dynamics is a well established simulation technique [50–61], with the University of Houston Brownian Dynamics program "UHBD" being the most well-known implementation [62]. Recently BD has had something of a renaissance and is being used for increasingly large-scale simulations, approaching the levels of simulation required for e.g. signalling pathways (see Figure 1.2).

To summarise the practical aspects of the algorithm, there are two key ingredients to BD: first is the diffusion tensor D, which controls the hydrodynamic motion of the protein; second is the total force represented by the term $-\nabla U(s(t))$. In practice the force usually comprises the long-range electrostatic force with an additional Lennard-Jones potential for very short-range interactions.

1.4 Electrostatics of biological molecules

This section describes the available methods for calculating electrostatic interactions using some of the representations of proteins in solution outlined above.

1.4.1 Basic electrostatics: Coulomb's Law & Dielectric Constant

The Electric field, E, as a function of position r from a point charge Q is given by Coulomb's Law (Equation 1.8), $\hat{r}$ is the unit vector pointing from the charge to the evaluation point,

and ε is the permittivity of the medium.

$$E(r) = \frac{Q}{4\pi\varepsilon|r|^2}\hat{r} \tag{1.8}$$

The permittivity ε can be expressed as the product of $\varepsilon_0\varepsilon_r$ where ε_0 is the permittivity of free space (a constant with value approximately $8.854 \times 10^{-12} m^{-3} kg^{-1} s^4 A^2$) and ε_r is the relative permittivity, also known as the dielectric constant, of the medium, which is 1 for a vacuum, and greater than 1 for all other materials. The electric field is the force per coulomb of charge which a 'test charge' at that point would experience[7].

The relative permittivity is a number which models the polarization of the medium due to the electric field: it is the bulk parameter arising from a combination of microscopic physical effects. The smallest polarization effect is the displacement of the electronic charge distribution surrounding atoms, which creates a slight dipole effect, reducing the net electric field; electronic polarization in proteins is modelled by a dielectric constant of about 2.0. Polar molecules (i.e. those with a permanent dipole arising from asymmetry in electronic distribution due to chemical bonding) which are free to rotate in the applied field have an even greater polarization effect; for example water (at 20°C) has a relative permittivity (dielectric constant) of 78, arising from dipole reorientation in the vicinity of a charge.

Materials with a more complicated structure, such as proteins which are composed of what could be considered a low dielectric core (i.e. rigidly bonded atoms, subject only to electronic polarization) with outer regions of higher polarizability corresponding to flexible polar side-chain groups. Considering that proteins in the context of this thesis are generally assumed to be large molecules in solution, it is difficult to justify applying a single bulk material parameter such as dielectric constant to the entirety of a macromolecule like a protein; the exact meaning of the dielectric constant in this case depends on position within the protein, and in truth probably cannot be properly modelled without resorting to a QM approach. Nonetheless the "dielectric constant" ε_r of proteins is commonly assigned to be in the range 2-20, intended to account for the combination of electronic polarization of the core of the protein and some degree polarization produced by side-chain flexibility on the exterior. There is no "correct" value.

The electric field is related to the Electric Displacement field[8], D, by Equation 1.9 (for a linear, isotropic, homogeneous material):

[7] More precisely, it is the force per coulomb as the test charge magnitude disappears to zero, since the test charge itself would affect the field at that point if it had any magnitude at all.

[8] The electric displacement field is usually denoted D, so we follow that convention here, despite our previous usage of the letter D to represent diffusion tensor. We will not be referring to the diffusion tensor again in this chapter.

$$D(r) = \varepsilon E(r) \tag{1.9}$$

From Equation 1.9 the dielectric constant ε can also be interpreted as the degree to which electric flux lines (which make up the electric field E) are concentrated when a given medium is placed in an electrostatic field. D is a useful quantity as the normal component is continuous across dielectric boundaries.

Coulomb's law is a special case of the more general Gauss's Law (Equation 1.10) which in vector form relates the divergence of the electric field to the charge density (ρ).

$$\nabla \cdot E(r) = \frac{\rho(r)}{\varepsilon} \tag{1.10}$$

In the absence of a varying magnetic field, the electric field can be expressed as equation 1.11:

$$E(r) = -\nabla\phi(r) \tag{1.11}$$

where ϕ is a scalar potential field. Equivalently, the potential ϕ at a point is the Coulomb energy required to bring a test charge from infinity to that point (assuming that the test charge itself is infinitesimal so that it doesn't alter the potential field). Substituting equation 1.11 in equation 1.10 we can obtain Poisson's equation (Equation 1.12).

$$\nabla^2\phi(r) = \frac{-\rho(r)}{\varepsilon} \tag{1.12}$$

For the more general case of non-constant dielectric constant throughout the volume, i.e. dielectric constant is a function of position r, $\varepsilon(r)$, Poisson's equation can be expressed as Equation 1.13:

$$-\nabla \cdot (\varepsilon(r).\nabla\phi(r)) = \rho(r) \tag{1.13}$$

1.4.2 Debye-Hückel Theory

Since proteins natively inhabit a salty environment, the effect of salt on protein stability and kinetics cannot be neglected. Weakly ionic solutions of the type encountered in biology have been described by Debye and Hückel [63]. The net effect of ions in solution is similar to a polarization effect; the ions are free to migrate and will tend to adopt the lowest energy

configurations: negative charges will migrate to places of positive potential, and positive charges in regions of negative potential. At thermal equilibrium, the energy distribution of ions in solution will follow the Boltzmann distribution.

Using the results of Debye and Hückel it is possible to write an expression (Equation 1.14) for a quantity known as the "inverse Debye screening length" (units $\mathring{A}^{-1}$), denoted κ, which links the concentration of ionic solutions to the screening effect they have on the potential due to a point charge by a simple modification to the Coulomb Law (Equation 1.15).

$$\kappa = \sqrt{\frac{2N_A I q^2}{\varepsilon k_B T}} \tag{1.14}$$

where q in this equation is the proton charge; N_A is Avogadro's number (6.022×10^{23}); I is the ionic strength (concentration of the electrolyte scaled by charge squared) in mol/m^3; $\varepsilon = \varepsilon_0 \varepsilon_r$ is the permittivity of the solvent; $k_B T$ is the Boltzmann factor multiplied by absolute temperature: $k_B T = 2.48\, kJ/mol$ to 3 significant figures, at 298K.

$$\phi(r) = \frac{Q.e^{-\kappa|r|}}{4\pi\varepsilon|r|} \tag{1.15}$$

Equation 1.15 is commonly referred to as the "screened Coulomb potential", or the "Yukawa potential". Thus the Debye screening length κ^{-1} can be interpreted as the distance of electrolyte required to reduce the effective potential of a charge by a factor of e. For 100mM NaCl this distance is 9.74Å, which suggests that electrostatic interactions between macromolecules at physiological salt (typically a few hundreds of mM salt concentration) could in fact be less "long-ranged" than might otherwise be expected.

For a solution containing multiple ionic species, i, with individual valences z_i and bulk concentrations c_i, Equation 1.14 can be re-written as Equation 1.16 (for the most general case of a position-dependent dielectric $\varepsilon(r)$):

$$\kappa^2(r) = \sum_i \frac{q^2 z_i^2 N_A c_i}{\varepsilon(r) k_B T} \tag{1.16}$$

1.4.3 Explicit Solvent Models

In explicit solvent models such as MD, the electrostatic contributions can be calculated relatively simply through a Coulomb summation over all of the simulation space; all atoms are represented and the force field defines the partial charge on each atom given its chemical environment, so the location of all charges is known. However a direct Coulomb sum scales with $\mathcal{O}(N^2)$ which clearly becomes prohibitive for the large number of atoms within an

MD simulation of a solvated protein. Simplification methods are sometimes employed, for example a cutoff-radius beyond which electrostatics are assumed to be zero or a reaction-field approximation analogous to that for implicit solvents (discussed later in this chapter). The favoured choices for evaluation of the electrostatic potentials in current MD programs are the Particle-Mesh Ewald (PME), or Particle-Particle Particle-Mesh (P3M) methods, which are approximations to the direct Coulomb summation, rather than simplifications of the electrostatic model itself.

The dielectric behaviour of space within an MD simulation is generally treated as vacuum (i.e. dielectric constant of 1), which makes no attempt to account for local electronic polarization by the charges in the system. Polarizable force fields have been under development for more than two decades (they are mentioned as being under development in the 1990 review by Davis and McCammon [64]), and are now becoming available as alternatives to the more established force fields [65–67].

Note that the other sources of polarization (through rotation of polar groups, or rotation of water molecules) should be generally well modelled by MD since the simulation will allow groups to rotate relatively easily and the lowest energy configurations should naturally be well sampled.

1.4.4 Implicit Solvent Models

At this point it should be clear that large scale molecular simulations using current computational resources require an implicit solvent approach. Replacing water with a high dielectric continuum reduces the set of possible methodologies considerably. A simple illustration of the implicit solvent continuum model is given in Figure 1.3, which shows a hypothethical molecule containing a set of atomic partial charges of varying radii, embedded in a low dielectric volume (ε_{int}) and surrounded by a high dielectric solvent region (ε_{ext}) containing mobile ions.

The necessity of representing solvent implicitly led to much effort and progress in the field of continuum electrostatics in the 1980's [68–73], building on the preceding work of Born, Debye, Hückel and Kirkwood who examined such systems earlier in the 20th century. Most of the implicit solvent models in use today are based in some way on equations and methods which have been well known for many decades. Nonetheless, as evinced by the abundance of literature on this subject, the problems in applying the implicit solvent model to biological macromolecules remain of great interest. The following sections review the most important terminology, models and methods for implicit solvent protein electrostatics.

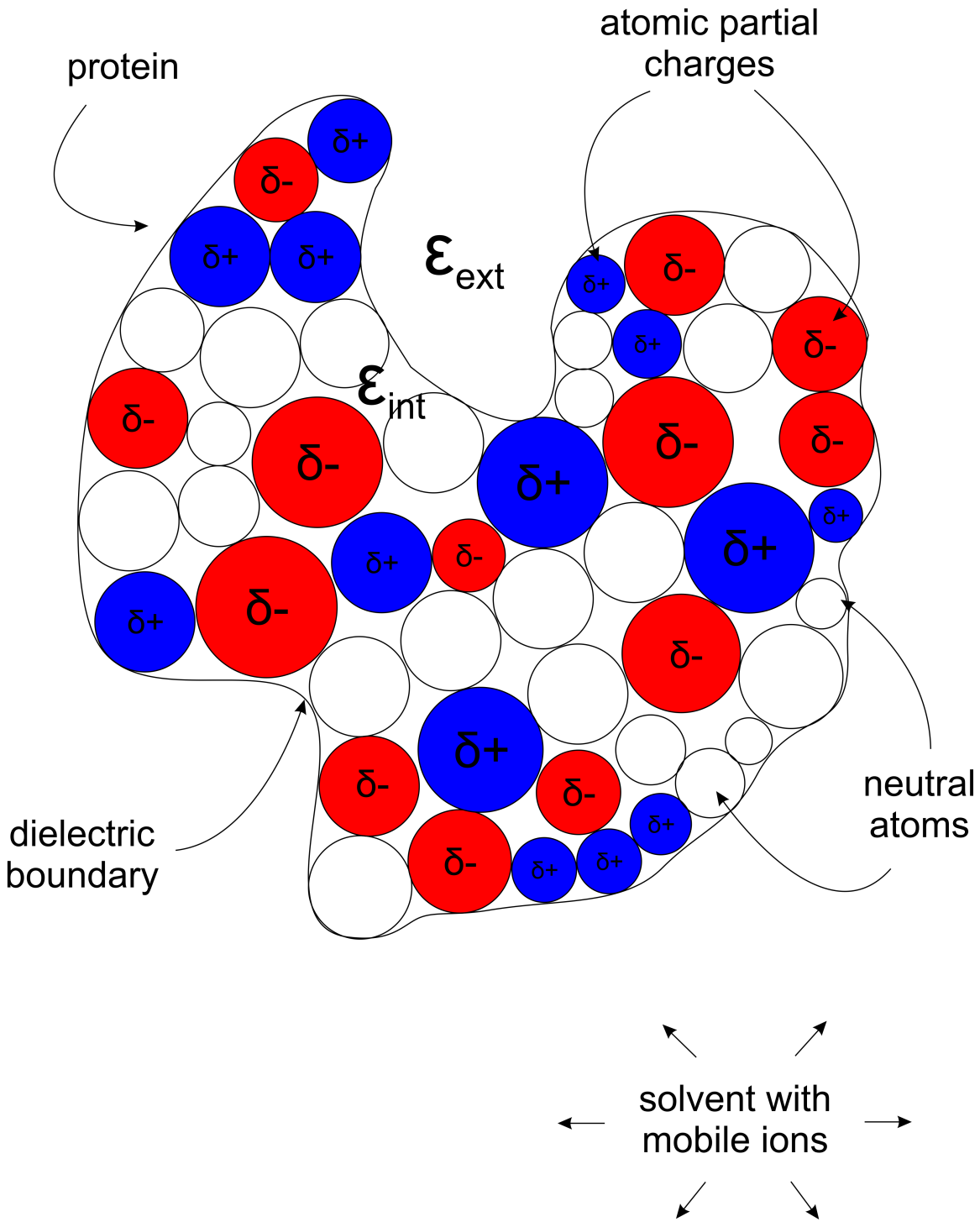


Figure 1.3: General model of a protein in water using implicit solvent: the solvent is represented by a high dielectric continuum (with mobile ions), while the protein is modelled as a low dielectric volume containing atoms carrying partial charges.

1.4.4.1 Solvation energy

A commonly encountered phrase in protein electrostatics is the solvation energy: that is, the energy difference between the solvated system in Figure 1.3 and the same protein surrounded by homogeneous dielectric ε_{int} rather than solvent.

The total solvation energy, ΔG_{solv}, is actually composed of several parts: the electrostatic (polar) solvation energy ΔG_{elec}; the van der Waals energy ΔG_{vdW}; and the non-polar component ΔG_{cavity} associated with the exclusion of the solvent from the dielectric "cavity" (Equation 1.17).

$$\Delta G_{solv} = \Delta G_{vdW} + \Delta G_{cavity} + \Delta G_{elec} \tag{1.17}$$

The electrostatic component of the solvation energy ΔG_{elec} is given by the effect of polarization produced within the solvent by the atomic partial charges within the protein, whilst the non-polar component ΔG_{cavity} incorporates an energy associated with the dielectric surface itself (a surface tension) as well as a more loosely defined entropic term which is due to the rearrangement of water molecules caused by the presence of the cavity. Finally the the term ΔG_{vdW} represents the change in van der Waals forces between the two states.

It has been noted that experimental solvation energies tend to scale more or less linearly with surface area, suggesting that the non-polar term can be modelled by a linear scaling of the solvent accessible surface area (SASA) [74]. However recent studies by the Baker group have suggested that a more complicated formulation is necessary [75]. The result is that fitting the electrostatic component of solvation from experimental data (i.e. parameter tweaking the electrostatic model) is complicated by the fact that the non-polar contribution is non-trivial as well.

Whilst parameterisation is generally carried out against the solvation characteristics of small molecules (e.g. amino acids), because it is difficult to obtain experimental solvation energies for whole proteins. Nonetheless there has been some effort to validate solvation energy calculations through e.g. "blind trials" against small molecules [76], or against explicit water MD [77].

The values of electrostatic solvation energy, ΔG_{solv}, are not of any particular use in themselves as they are somewhat arbitrary energies relative to a hypothetical state (a uniform dielectric). However, insofar as the change in electrostatic solvation energy represents the interaction between a protein and the surrounding solvent, the *change* in this energy term ($\Delta\Delta G_{solv}$) as a result of changes in protein structure, or the solvent environment, can be of use in simulating the behaviour of proteins in solution. In such cases the non-polar and van der Waals terms must be included separately. For our purposes we assume that it

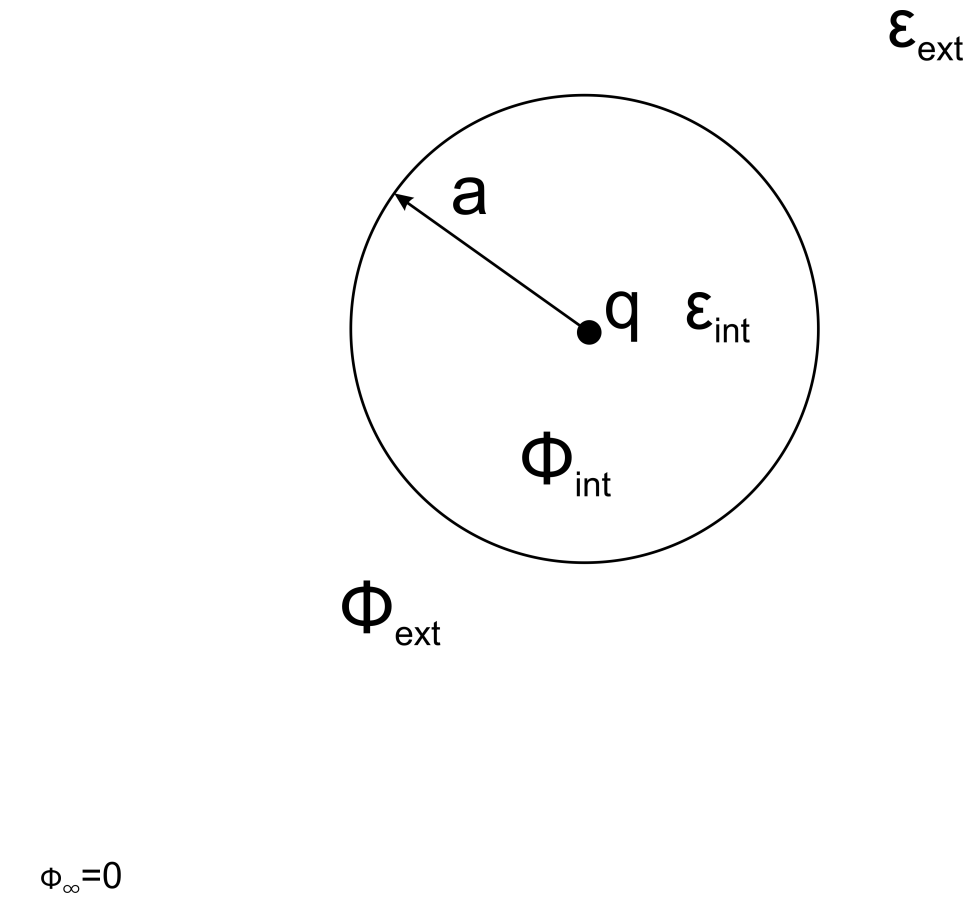


Figure 1.4: The Born ion: a spherical cavity of radius a, centre on a charge q (which we can assume for convenience is located at the origin).

is the electrostatic component of solvation energy which is of interest, and we do not make any further mention of the other solvation terms in this work.

1.4.4.2 The Born Ion

Perhaps the simplest implicit solvent model is the Born model of an ion [78] as illustrated in Figure 1.4. The ion can be considered a spherical cavity of radius a, with internal dielectric denoted ε_{int} and external dielectric ε_{ext}, and a point charge q at the centre[9]. A real protein does not correspond very closely to the Born ion model, either in surface geometry or charge distribution, but the results of this simple model may give us some insight into how to go about building a more complicated and realistic model[10].

We mentioned above that the (electrostatic) solvation energy is of use in quantifying the effects of the solvent on a macromolecule, so we will now derive the Born solvation energy,

[9]Reminder: the dielectric constants are the relative permittivities; the permittivity of free space ε_0 we denote explicitly in the formulae, which are in SI units.

[10]It is also one of the few systems for which it is possible to obtain an analytic solution.

ΔG_{Born}, which is defined as the energy change in bringing the ion from a uniform dielectric into the solvated environment. In order to calculate this, we will begin by trying to calculate the *total* value of the energy of the system (G_{elec}), which requires determining the value of potential (ϕ_{int}) at the location of the charge.

We begin with the integral form of Gauss' Law (Equation 1.10), which states that the flux of the electric displacement field (D) leaving a surface is equal to the total charge contained within the volume, and using the relation $D(r) = \varepsilon(r)E(r)$ (Equation 1.9) we can write Equation 1.18, which is this relation applied to some arbitrary spherical shell of radius r which may be inside or outside of the surface defined by radius a in Figure 1.4.

In this case, since the system is spherically symmetric, we can treat the integrand $\varepsilon(r)E(r)$ as a one-dimensional function of the radius from the centre, which is a constant value over the surface, which makes the value of the integration simply the surface area of a sphere of radius r.

$$\oint_S \varepsilon(r)E(r) \cdot dA = q \tag{1.18}$$

$$\varepsilon(r)E(r) \oint_S dA = q \tag{1.19}$$

$$E(r) = \frac{q}{4\pi|r|^2\varepsilon(r)} \tag{1.20}$$

This is a general formula for electric field, E, (negative gradient of the potential ϕ) as a function of radius r. In order to obtain a formula for the potential at any point outside the cavity, $\phi_{ext}(r)$, $r \geq a$, we can assume that the application of Gauss' Law was carried out on a spherical shell outside of the cavity, in the external dielectric. Integrating Equation 1.20 in the external region gives:

$$\phi_{ext}(r) = \frac{q}{4\pi r\varepsilon_0\varepsilon_{ext}} + C_1 \tag{1.21}$$

In order to find the constant of integration C_1 we impose the boundary condition that $\phi \to 0$ as $r \to \infty$; therefore $C_1 = 0$.

Within the cavity, the potential $\phi_{int}(r)$, $r \leq a$ is given by:

$$\phi_{int}(r) = \frac{q}{4\pi r\varepsilon_0\varepsilon_{int}} + C_2 \tag{1.22}$$

At the boundary $r = a$ the boundary condition $\phi_{int} = \phi_{ext}$ allows us to solve for C_2:

$$C_2 = \frac{q}{4\pi a\varepsilon_0\varepsilon_{ext}} - \frac{q}{4\pi a\varepsilon_0\varepsilon_{int}} = \frac{q}{4\pi a\varepsilon_0}\left(\frac{1}{\varepsilon_{ext}} - \frac{1}{\varepsilon_{int}}\right) \tag{1.23}$$

If the dielectric response is linear, the *total* electrostatic energy is given by:

$$G_{elec} = \frac{1}{2}\int_V \phi\rho dV = \frac{1}{2}q\phi_{int}(r_q) = \frac{q^2}{8\pi r_q\varepsilon_0\varepsilon_{int}} + \frac{q^2}{8\pi a\varepsilon_0}\left(\frac{1}{\varepsilon_{ext}} - \frac{1}{\varepsilon_{int}}\right)\ r = 0 \tag{1.24}$$

where r_q is the radius at which the charge is located. With q defined as a point charge at the origin ($r_q = 0$), this solution leads to an infinite potential and therefore infinite energy using this expression. Consequently the quantity G_{elec} is not particularly meaningful.

However we have already defined the Born solvation energy, ΔG_{Born}, as the *change* in energy between a reference uniform dielectric state and the solvated state, which can be found by finding the change in G_{elec} between those two states: conveniently the "self-energy" term cancels, leaving the relatively simple Equation 1.25.

$$\Delta G_{Born} = -\frac{q^2}{8\pi a\varepsilon_0}\left(\frac{1}{\varepsilon_{int}} - \frac{1}{\varepsilon_{ext}}\right) \tag{1.25}$$

1.4.4.3 The Generalized Born (GB) Method

The Born ion gave us a simple relationship between the solvation energy and a charge in a spherical cavity. However within a real protein the atomic charges are not isolated spheres surrounded by high dielectric solvent, there is overlap between the spheres and not all of the atomic charges are exposed to the solvent. Although it is clearly not possible to use Equation 1.25 to describe a protein, the Generalized Born model, introduced by Still *et al.* in 1990 [79] is an attempt to derive a functional form which combines a more realistic representation of the protein structure and inter-atomic electrostatic energy terms, with the simplicity of the functional form of the analytic Born ion.

A simple derivation of the pair-wise Generalized Born method as summarised by Bashford and Case [74] is as follows. Consider a number n of widely separated spherical charges q (separations between the i^{th} and j^{th} charges denoted r_{ij}) (as illustrated in the middle picture of Figure 1.5 for four such charges), with radii a_i and dielectric constants ε_{int} and ε_{ext}.

The change in total electrostatic solvation free energy, ΔG_{es}, between the reference state[11] at the top of Figure 1.5 and the heterogeneous "molecule" in the middle picture of Figure 1.5 is the sum of the change in Coulomb interactions between each charge $\Delta G_{Coulomb}$,

[11] commonly the reference state is assumed to be a vacuum; i.e. $\varepsilon_{ext} = \varepsilon_{int} = 1$

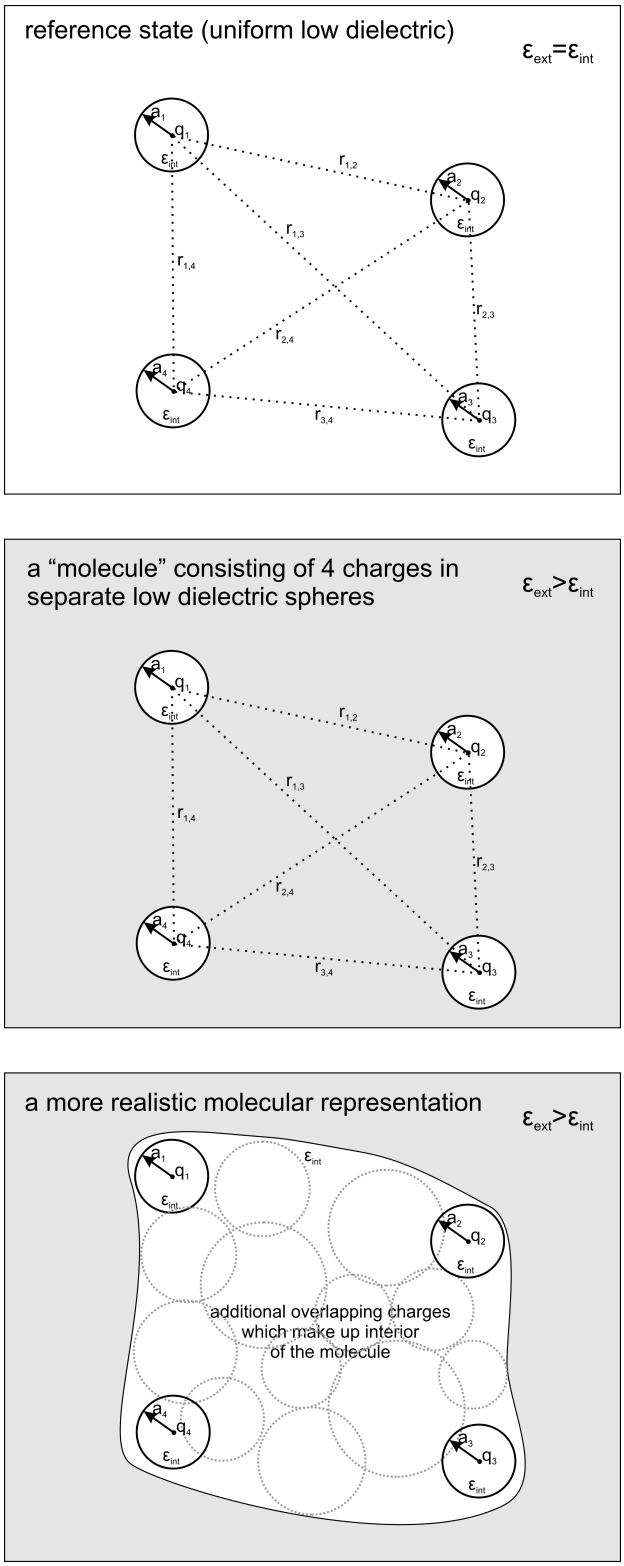


Figure 1.5: The Generalized Born model of a "molecule": (top) reference state with uniform dielectric; (middle) heterogeneous dielectric model for well-separated charges, for which Equation 1.26 is exact; (bottom) more realistic model containing additional overlapping charges: this is described exactly by the Poisson equation, but can also be approximated by the Generalized Born equation (Equation 1.27).

plus the Born solvation free energies of the individual charges ΔG_{Born} (assuming they are well separated, the Born ion model of an isolated cavity will be a good approximation for each charge).

This can be written as Equation 1.26:

$$\begin{aligned} \Delta G_{es} &= \Delta G_{Coulomb} + \Delta G_{Born} \\ &= \left[\left(\frac{1}{\varepsilon_{ext}} - \frac{1}{\varepsilon_{int}}\right)\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{q_i q_j}{4\pi r_{ij}\varepsilon_0}\right] - \left[\sum_{i=1}^{n}\frac{q_i^2}{8\pi a_i \varepsilon_0}\left(\frac{1}{\varepsilon_{int}} - \frac{1}{\varepsilon_{ext}}\right)\right] \end{aligned} \quad (1.26)$$

Equation 1.26 is exact and correct for the middle picture of Figure 1.5 and has a functional form which is simple and whose components are simple to attribute to physical quantities. However the separate charges do not closely resemble a real molecule. A more realistic model of a molecule is shown in the bottom picture of Figure 1.5: here the atoms are part of a single low dielectric volume containing a larger number of overlapping atoms, rather than 4 isolated charges.

The aim is to move from Equation 1.26 for separate charges/spheres to an approximate expression which models the solvation energy of overlapping atoms in a "real" molecule. Using the functional form of Equation 1.26 as inspiration and collecting "similar-looking" terms into a single expression results in the Generalized Born equation [79], Equation 1.27 , which resembles something like a cross between the change in Coulomb energy and the analytic Born solvation formula:

$$\Delta G_{es} \approx -\frac{1}{8\pi\varepsilon_0}\left(\frac{1}{\varepsilon_{int}} - \frac{1}{\varepsilon_{ext}}\right)\sum_{i=1}^{n}\sum_{j=i}^{n}\frac{q_i q_j}{f_{GB}} \quad (1.27)$$

There is no suggestion that Equation 1.27 is mathematically equivalent to Equation 1.26: there is a double summation over all points, and an arbitrary function f_{GB} has been introduced! The function f_{GB} controls the pairwise energy contribution between charges, and appears in Equation 1.27 as a "Born-radius-like" term controlling the extent to which a particular pair of charges behaves as a classical Born ion. There is no unique definition for the functional form of f_{GB}, but Still *et al.* suggest the following:

$$f_{GB} = (r_{ij}^2 + R_{ij}^2 e^{-D})^{\frac{1}{2}} \quad (1.28)$$

where: $R_{ij} = \sqrt{R_i R_j}$ and $D = \frac{r_{ij}^2}{4R_i R_j}$. The effective Born radii R_i are parameters of the model which are based on the actual radii of the atoms, a_i, modified to account for local effects and extent of burial.

This function provides a blending/interpolation in which pairs of charges have an effective Born radius related to their actual radii (somewhat indirectly via R_i) when close together, and controlled by their separation when far apart; thus pairs of charge in close proximity have a strong effect on each others' solvation energy contribution, whilst distant charges have much smaller energetic contributions.

According to Still *et al.* this particular choice of function (with $R_i = a_i$) accurately reproduces the Born solvation energy for combined charges when the separation r_{ij} tends to zero; the Onsager reaction field energy for a dipole within a spherical cavity (to within 10%) when separation is less than one tenth of the effective radius a_{ij}; the Born + Coulomb interaction energy for separate spheres when the separation $r_{ij} > 2.5a_{ij}$ (within 1%).

In the 20 years since Still *et al.* suggested the GB model, various enhancements and refinements have been suggested to account for e.g. salt effects [74, 80–82], and many variants of the original formulation exist.

Choice of effective Born radii, R_i From the above it should be clear that the output of the GB model depends very strongly on the parameters used for the radii of the atoms of a protein[12], as well as the charges assigned to them. Ideally, R_i should be set such that the analytic Born energy using that radius matches the self-energy of the charge in its reaction field as determined by solving the Poisson equation for the protein with only atom i charged, but with the dielectric boundary conditions of the full protein applied. This per-atom parameterisation would give a perfect match between GB and PB methods for a single set of dielectric and solvent conditions, but is obviously impractical as it involves all of the work of both methods. A comparison of the relative performance between GB and PB methods is described by Feig *et al.* [83].

[74] outline various methods for choosing R_i analytically based on assumptions of overlapping spheres, with corrections to account for the degree of overlap, which are typically derived empirically in order to fit experimental values. Alternatively surface integration can be used to find suitable values for R_i as described by [84].

In summary GB and its related refinements offer a relatively fast computational method for solving the problem of protein electrostatics, however the method relies heavily on empirical fitting rather than following any particularly robust model of the physical reality. GB is sometimes used within MD to provide a very quick and simple estimate for electrostatic interactions (giving faster evaluations, so more timesteps per hour of CPU-time than

[12] though according to Still *et al.* the solvation energy is more sensitive to the relative positions than the radii themselves; thus uncertainty in atomic position limits the required accuracy of the parameterisation of the effective Born radii.

is obtained for more detailed electrostatic calculations). However for "production runs" GB is very seldom used as PME and P^3M offer much higher accuracy.

1.4.4.4 Inducible Multipole Models

Langevin dipoles The inducible dipole model described by Warshel and Levitt [28] (also well summarised in [85]) models the solvent as a grid of inducible point dipoles (also known as Langevin dipoles) which represent the polarization of the solvent by the source charges. The protein is modelled as a set of point charges, plus point dipoles to represent electronic polarization of the protein. The magnitude of each point dipole is found by assigning a polarizability to that point and applying the electric field produced by the source charges to the induced dipoles, creating a reaction field of dipoles. This process is repeated iteratively, including the effect of the dipoles on the total field until the values of the dipoles are converged.

The advantage of the method is that the polarization can be selected to adopt physically plausible values at each point, avoiding the use of a macroscopic dielectric constant which, it can be argued, is not appropriate for proteins on the microscale [65, 86].

The model can be classed as implicit solvent in that the water is replaced by a grid of dipoles rather than explicit water molecules with partial charges; however the requirement to fill the volume with dipoles to represent the solvent makes the method unsuitable for very large scale simulations.

Inducible Multipole Solvation model The Inducible Multipole Solvation (IMPS) model described by Davis [87] is similar in spirit to the Langevin dipoles model. Point multipoles (i.e. a linear combination of point dipoles, quadrupoles etc.) are placed at charge centres in the protein; the high dielectric solvent and low dielectric protein are replaced by a homogeneous high dielectric continuum. The inducible point multipoles (whose values are related to the electric field produced by the source charge distribution) are intended to reproduce the effect of the dielectric discontinuity which has been removed.

1.4.4.5 The Test-Charge Approximation

The "test charge approximation" is not a model of electrostatics *per se*, but an approach by which the interaction between two proteins can be estimated, for use in a simulation where forces or energies of interaction are required. The term "test charge" is normally used in association with the physical interpretation of an electric potential: the potential is the energy per unit charge which would be attained by an infinitesimal charge brought into that

region, such that it did not affect the potential. Here a similar meaning applies, but on the scale of proteins.

Firstly the values of potential around a protein, in a grid extending to some distance from the surface (beyond which the potential is assumed to be negligible; with ionic screening this may be a realistic approximation only a few tens of Angstroms away), are found by some method e.g. through solving the Poisson-Boltzmann Equation (described in Section 1.4.4.7) for the isolated protein. The premise of the test charge approximation is that another (target) protein moving into the "source" potential field does not significantly perturb it, and the charges which make up the target protein can be considered to interact directly with the source field; thus the total force can be found by multiplying target charges by the interpolated source electric field where they overlap:

$$\Delta G_{interaction} = \sum_k q_k \phi$$

$$F = \sum_k -q_k \nabla \phi$$

The advantage of the method is that the potential grids of the proteins can be pre-computed, and the charge-grid interactions can be carried out rapidly, making the method computationally much faster than obtaining a full solution to the PBE at each timestep.

The disadvantage is that the model is somewhat physically implausible as it entirely neglects the effect of the target protein on the electric field of the source protein, both in terms of charge distributions and dielectric effects. The missing dielectric effects are twofold: firstly the presence of the low dielectric volume represented by the target protein will affect the shape of the electric field produced by the source protein, through mutual polarization effects. Secondly the extent to which charges in the low dielectric target protein (that is, the "test charges") experience the electric field is underestimated by the fact that the source field is calculated for solvent dielectric, whereas the test charges exist in a low dielectric medium.

Furthermore the test charge model may significantly underestimate the energy for charge interactions when the solvent contains mobile ions, because the value of potential in the source protein field will include a salt screening effect, whereas the actual environment of the charges in the target protein is ion-excluded: the potential at the location of the test-charge according to the source protein will be much too low.

1.4.4.6 Effective Charges for Macromolecules (ECM)

Effective charges for macromolecules in solution (ECM) [88] is an attempt to improve upon the test charge approximation by reducing the salt-dependent error, and at the same time reincorporating an estimate for one of the missing dielectric effects.

ECM scales the charges of the "target" protein such that the (screened) Coulomb potential due to the naked charges in solution closely approximates the "actual" potential of the full heterogeneous dielectric model[13]. The potential produced by the charges in high dielectric solution will be lower than the potential fields they produce when in low dielectric protein (very much so, in the case of ionic solution), so this scaling will be positive: the charges will be magnified by some factor. The augmented charges are then used as per the "test-charge" method, in that they are multiplied by the pre-calculated potentials around a source protein to obtain a total energy (or force). It is the intention of the method that the degree to which the source potential underestimates the potential (due to ionic screening) is counterbalanced by the scaling of the ECM charges, resulting in energies which are more accurate to those which would be obtained through more rigorous methods (such as solving the Poisson-Boltzmann Equation).

The choice of scalings for charges is obviously critical to the success of the method. The charges are enlarged such that there is a least-squares fit between the potentials in a grid produced by the ECM method and the "actual" potentials. The scaling automatically accounts for the effect of the low dielectric environment of the target protein as well as the solvent effect; however the effect of the low dielectric source protein and it's collection of charges on the potential field of the source protein remains unaccounted for. (This method slightly resembles the IMPS model (described above, Section 1.4.4.4) in that the dielectric boundary has been removed, and a correction of the source charges applied to approximate the effect of the missing low dielectric; in ECM the charges are scaled in order to "best-fit" the true potential; in IMPS, point multipoles are added but without any attempt at "fitting" the potential).

Gabdouilline and Wade [88] showed that for bi-molecular systems (e.g. protein-ligand interactions) the energies derived using ECM approach those found by complete solution of the PBE, and are certainly much closer than those found using the test charge approximation. In addition, there are various possible treatments and optimizations possible to "regularize" the effective charges (this helps avoid overfitting).

Rather than dealing with all of the charges within a protein the user can choose to place effective charges at the alpha-carbon of amino acids, reducing the number of charges required

[13] Found, for example, by solving the Poisson-Boltzmann Equation (see Section 1.4.4.7) for the heterogeneous case of protein charges embedded in a low dielectric material, as illustrated by Figure 1.3.

to represent a protein. Additionally the potential grid used to fit the scaled charges can be chosen to be an "average" set of potentials from multiple protein conformations, which is an appealing method for dealing with minor conformational variations of the protein.

In summary the ECM method allows rapid computation of protein interactions in solution, with improved accuracy compared to the test charge approximation. The ECM method has been used in many Brownian Dynamics simulations [52, 53, 56, 58, 89], giving seemingly good results, and is the electrostatics model adopted by McGuffee and Elcock in their impressively large simulations of crowded protein systems [60, 61].

However it is not clear that the ECM model, which appears to be a good approximation for bimolecular interactions, is a good approximation for the more complicated many-body interactions found in crowded systems, and particularly for interactions between large proteins where the dielectric effects are expected to be more significant. It seems likely to us that the presence of large numbers of macromolecules, which are not only regions of low dielectric but also ion excluded, will significantly distort the "infinite dilution" potential which ECM assumes exists around each macromolecule. In order to take account of these effects, a more rigorous approach to solving the implicit solvent system of Figure 1.3 is required.

1.4.4.7 Poisson-Boltzmann (PB) Methods

The Poisson-Boltzmann Equation (PBE) The ionic charge distribution within the solvent can be incorporated into the Poisson equation as an additional source of charge density $\rho_{mobile}(r)$, Equation 1.29:

$$-\nabla \cdot (\varepsilon(r).\nabla\phi(r)) = \rho_{fixed}(r) + \rho_{mobile}(r) \tag{1.29}$$

Unlike the source charge distribution which is known at the outset, the ionic charge distribution responds to the electric field and relocates accordingly. If we assume that the ions behave as a large population of point charges which follow the Boltzmann distribution with respect to their charge-potential interaction[14], we can write the ionic charge density term as:

$$\rho_{mobile}(r) = \sum_i q z_i c_i e^{-\frac{1}{kT}(q z_i \phi(r) + \lambda(r))} \tag{1.30}$$

where the sum is over each ionic species, i, in the system; z_i is the valency of the ion, c_i is the concentration of the ion in the bulk, and the potential function $\lambda(r)$ controls the accessibility of ions to specific regions of space (i.e. in the solvent this function has a value of

[14] this neglects any other energy terms for the ions, e.g. desolvation effects of the ions

zero, whereas in the solute an infinite value of this function corresponds to a zero probability of finding ions there)[15]. Note that $\lambda(r)$ may or may not match the dielectric boundary; if we model a Stern Layer around the protein where no ions are permitted, then the ion exclusion function is not the same as the dielectric boundary. In addition this function may be different for each ionic species, as in general the ionic radius will vary.

Combining Equations 1.29 and 1.30 gives the Poisson-Boltzmann Equation (PBE), Equation 1.31:

$$-\nabla \cdot (\varepsilon(r).\nabla\phi(r)) = \rho_{fixed}(r) + \sum_i q z_i c_i e^{-\frac{1}{kT}(q z_i \phi(r) + \lambda(r))} \tag{1.31}$$

This is a general description for steady-state continuum electrostatics: the validity of the equation is only limited by the treatment of dielectric constant throughout the volume, and by the concentration of ions at which the assumption of no significant ion-ion interaction becomes violated. However the PBE in this form is generally non trivial to solve because the potential appears in the exponential term on the right hand side.

The PBE can be linearised by making further assumptions about the nature of the ionic interactions:

1. All ionic species have same exclusion function with respect to the solute (i.e. approximately the same radii)
2. The magnitude of the ion energy is much smaller than kT (i.e. $q z_i \phi(r) \ll kT$)
3. The bulk solution is electroneutral

Using assumption #1 allows us to factorize $\lambda(r)$ out of the summation to give Equation 1.32:

$$\rho_{mobile}(r) = e^{-\frac{\lambda(r)}{kT}} \sum_i q z_i c_i e^{-\frac{1}{kT}(q z_i . \phi(r))} \tag{1.32}$$

Assumption 2 allows us to approximate the ionic charge-potential term using a Taylor expansion of $e^x = 1 + x + \frac{x^2}{2} + \ldots$ where $x = -\frac{1}{kT}(q z_i \phi(r))$ and, neglecting the quadratic terms onwards, gives Equation 1.33:

$$\rho_{mobile}(r) = e^{-\frac{\lambda(r)}{kT}} \sum_i q z_i c_i \left[1 - \frac{q z_i \phi(r)}{kT}\right] \tag{1.33}$$

[15]The term kT throughout the remainder of the chapter is the thermal energy term, $k_B T$ (encountered previously) but we have omitted the subscript B from the Boltzmann constant k_B to improve legibility.

Assumption #3 of bulk electroneutrality ensures that the summation $\sum_i q z_i c_i = 0$, which removes the first term in square brackets, leaving Equation 1.34:

$$\rho_{mobile}(r) = -e^{-\frac{\lambda(r)}{kT}} \sum_i \frac{q^2 z_i^2 c_i}{kT} \phi(r) \tag{1.34}$$

Using Equation 1.16 and defining a more convenient ion-exclusion function $V(r) = e^{-\frac{\lambda(r)}{kT}}$ such that $V(r)$ is 1 in the solution and 0 in the solute, allows us to write Equation 1.35:

$$\rho_{mobile}(r) = -V(r)\varepsilon(r)\kappa^2(r)\phi(r) \tag{1.35}$$

Combining Equations 1.29 and 1.35 and moving the mobile charge terms to the left hand side, leads to the linearized Poisson-Boltzmann Equation, Equation 1.36.

$$-\nabla \cdot (\varepsilon(r).\nabla\phi(r)) + V(r)\varepsilon(r)\kappa^2(r)\phi(r) = \rho_{fixed}(r) \tag{1.36}$$

Finite-difference Methods (FD) The first description of a numerical method to solve the PBE for biomolecular electrostatics was Warwicker and Watson [68]. The idea gained popularity rapidly, and a number of finite-difference solvers for the PBE now exist (for example APBS, Delphi, MEAD, UHBD to name a few popular implementations).

The general method of finite-difference solvers is to divide space into a volumetric grid; the solution (electric potential, ϕ) is found at each grid point by initializing the values, then iteratively refining the solution until a set of values at grid points is found which satisfies the Poisson-Boltzmann Equation. The dielectric boundary and ion-exclusion functions are discretized by the volumetric grid, such that the actual molecular boundary passes through some volumetric region between two grid points.

Recent implementations of the FD method for the PBE, e.g. APBS [90], use a multi-grid approach where approximate solutions are used as the boundary values for a finer grid iteration, leading to rapid convergence and better accuracy near the molecular boundary (since the rapidly-varying solution at the boundary can be carried out at the finest grid resolution, whilst the rest of the volume is dealt with more sparsely).

The extraction of physically meaningful forces and energies from finite difference PBE solvers can be non-trivial [91–93] as the quantities must be extracted by interpolation over grid points, which for forces or energies near the boundary may produce large errors in the output.

Boundary Element Method (BEM) The Boundary Element Method (BEM) is a general method for solving partial differential equations, and is well established as a numerical technique in engineering (e.g. fracture mechanics; acoustics). A major advantage over finite-difference or finite-element methods is the reduction of dimensionality afforded by working with surfaces instead of volumes. A BEM solver should scale better than a finite difference method for large-scale simulations.

Conceptually it can be simpler to think of the boundary element method as finding the surface charge distribution (so-called single and double layer charge densities) over the protein which (from the point of view of the interior charges) represents the polarization of the surrounding solvent by the protein; this is equivalent to solving the linearised Poisson-Boltzmann equation. The interaction between source charges in the protein and polarization charge density is equivalent to the electrostatic solvation energy.

The mathematical details of the BEM are given in full in Chapter 2. At this point we will merely note that the BEM in general requires discretizing the surface into elements (e.g. planar triangles) and solving for surface solutions of potential and electric field at points over that surface (e.g. the set of vertices of the planar triangles). Solving the BEM involves solving a linear system of equations $Ax = b$, where A can be written as a $2N \times 2N$ matrix in which each row represents the boundary integral equation for the value of a surface solution at one of the N surface points (either potential, ϕ, or the normal component of electric field, $E.n$)[16], as a function of the solutions at all other surface points.

The use of the BEM in protein electrostatics is not especially new and the principle was introduced more than 25 years ago by Zauhar [94] (although the term "boundary element method" does not explicitly appear in that article). The method was investigated further by various groups [95–110].

The initial drawback with the BEM was the scaling of the algorithm as $\mathcal{O}(N^2)$. This was solved by use of a far-field approximation, such as the Fast Multipole Method (FMM) [96, 111–113], or FFTSVD [106], which improve performance first to $\mathcal{O}(N \log N)$, and most recently to $\mathcal{O}(N)$ [103].

Other refinements have focused on the numerical stability of the boundary integral formulation: the simplest boundary integral formulation are Fredholm equations of the first type (e.g. the work of Boschitsch *et al.* [97] or the earlier work of Lu *et al.* [99], whose work is discussed in detail in Chapter 2) which leads to numerical instability within the BEM as the number of solution points increases. A more stable set of equations (Fredholm equations of the second type) were developed by Juffer *et al.* [114].

[16] N surface elements, 2 unknowns per element, hence the linear system has dimensions $2N \times 2N$.

The discretization of the protein surface into elements is central to the BEM. Planar triangles are a common choice due to their simplicity for numerical integration; however the use of curved elements has been investigated [106, 107]. Other possibilities include the "node-patch" method of Lu *et al.* [102] (which will also be discussed in Chapter 3). Another alternative, defining a parametric surface based on multipole expansions, was suggested by Kuo *et al.* [115]. Finally Tausch *et al.* [116] discuss some of the finer points of numerical integration over surface elements for the BEM.

Parameterisation Whether the PBE is solved by the finite-difference method or BEM, the parameterization of the dielectric boundary, choice of dielectric constant, and choice of partial charges/atomic radii has a large effect on the output of the PBE model. These issues are discussed in various places in the literature, such as the work of Swanson *et al.*[117, 118] Nina *et al.* [119, 120] and Yu, Geng and Wei [121, 122].

1.5 Thesis Overview

This chapter has introduced the motivation for large scale molecular simulation, and summarised the current methodologies available. We conclude that the most likely simulation method for simulations on the scale of intracellular signalling pathways is Brownian Dynamics, but note that to date the method misses that scope by several orders of magnitude.

In order to further the application of Brownian Dynamics into this space, it is necessary to have a compatible model for intermolecular forces (i.e. long-range electrostatics) which takes into account all of the mechanisms of interaction which we suggest are of interest to biology. The list of possible electrostatics modelling methodologies is short: we argue that the PBE is the only reasonable approximation for the crowded, salty, cytosol conditions we would like to be able to simulate.

This thesis introduces a state of the art Boundary Element Method (BEM) electrostatics solver for the linearised PBE called BEEP (Boundary Element Electrostatics Program), and gives results for the use of BEEP in biologically relevant problems.

Chapter 2 provides the mathematical basis for the solution of the linearised PBE with the BEM (in combination with a Fast Multipole Method (FMM)). Chapter 3 describes our implementation of this algorithm and presents results for analytic spherical test cases, whilst Chapter 4 discusses the parallelisation and performance of the BEM/FMM algorithms. In Chapter 5 we discuss the application of BEEP to proteins, and examine a bimolecular

protein-protein interaction to explore the capabilities and limitations of the BEM for biological macromolecules. Finally Chapter 6 discusses our plans for future work to extend and improve upon the methods detailed in this thesis.

Chapter 2

Mathematical Preliminaries: BEM & FMM

2.1 Outline of this chapter

This chapter summarizes the mathematical basis of the Boundary Element Method (BEM) for electrostatics, and how it can be used to solve the linearized Poisson-Boltzmann Equation (PBE) which we described in Section 1.4.4.7 of Chapter 1. We provide a detailed explanation of the Fast Multipole Method (FMM) and how it may be applied within the BEM to produce a method which has linear scaling with respect to number of surface elements used to describe a dielectric boundary.

2.2 The Boundary Element Method (BEM) for protein electrostatics

2.2.1 Derivation of the Boundary Integral Equations

The following derivation for the potential and its normal derivative are taken from the work of Lu *et al.* [99, 103]. We will start by restating the linearized Poisson-Boltzmann Equation which relates the electric potential (ϕ) as a function of position (r) to the fixed charge distribution (within the protein), taking into account the dielectric permittivities (henceforth referred to as dielectric constants) and the effect of ionic screening controlled by the inverse screening length κ (both functions of position).

$$-\nabla \cdot (\varepsilon(r).\nabla\phi(r)) + V(r)\varepsilon(r)\kappa^2(r)\phi(r) = \rho_{fixed}(r) \tag{2.1}$$

where $V(r)$ is a function which controls the regions where ions are permitted. Taking the ionic boundary to be the same as the dielectric boundary Γ, and the dielectric constants as shown in Figure 2.1, we write the PBE as it applies to each region, with the internal region defined by the volume Ω and the external region the complementary volume $\overline{\Omega}$. This yields Equation 2.2 for the internal potential ϕ_{int} and Equation 2.3 for the external potential ϕ_{ext}.

$$\begin{aligned} \nabla^2\phi_{int}(r) &= -\frac{\rho_{fixed}}{\varepsilon_{int}}, & r \in \Omega \\ &= -\frac{1}{\varepsilon_{int}}\sum_k q_k\delta(r-r_k), & r \in \Omega \end{aligned} \tag{2.2}$$

$$\begin{aligned} -\varepsilon_{ext}\nabla^2\phi_{ext}(r) + \varepsilon_{ext}\kappa^2\phi_{ext}(r) &= 0, & r \in \overline{\Omega} \\ \nabla^2\phi_{ext}(r) - \kappa^2\phi_{ext}(r) &= 0, & r \in \overline{\Omega} \end{aligned} \tag{2.3}$$

The Dirac delta functions $\delta(r-r_k)$ in Equation 2.2 model the fixed point charges of the protein: the charge density is zero except at the exact point $r = r_k$, where r coincides with one of the k point charges.

Green's Second Identity (for two general twice-differentiable scalar functions of position, say ζ and ψ) can be used to relate a volume integral of scalar fields to a surface integral over the boundaries of the volume (Equation 2.4):

$$\int_V \left(\zeta\nabla^2\psi - \psi\nabla^2\zeta\right) dV = \oint_S \left(\zeta\nabla\psi - \psi\nabla\zeta\right) dS = \oint_S \left(\zeta\nabla\psi\cdot n - \psi\nabla\zeta\cdot n\right) dA \tag{2.4}$$

where V and S are a volume and the surface bounding the volume respectively. The surface integral is a scalar product of the terms in brackets with the oriented surface area increment $dS = n(dA)$, n being the normal vector to the surface. The identity can be re-written in terms of the normal derivative $\frac{\partial}{\partial n}$:

$$\int_V \left(\zeta\nabla^2\psi - \psi\nabla^2\zeta\right) dV = \oint_S \left(\zeta\frac{\partial\psi}{\partial n} - \psi\frac{\partial\zeta}{\partial n}\right) dA \tag{2.5}$$

If we let the function ζ represent the potentials ϕ_{int} and ϕ_{ext} in Equations 2.2 and 2.3, we can use Green's second identity to rewrite the volume integrals as boundary integral

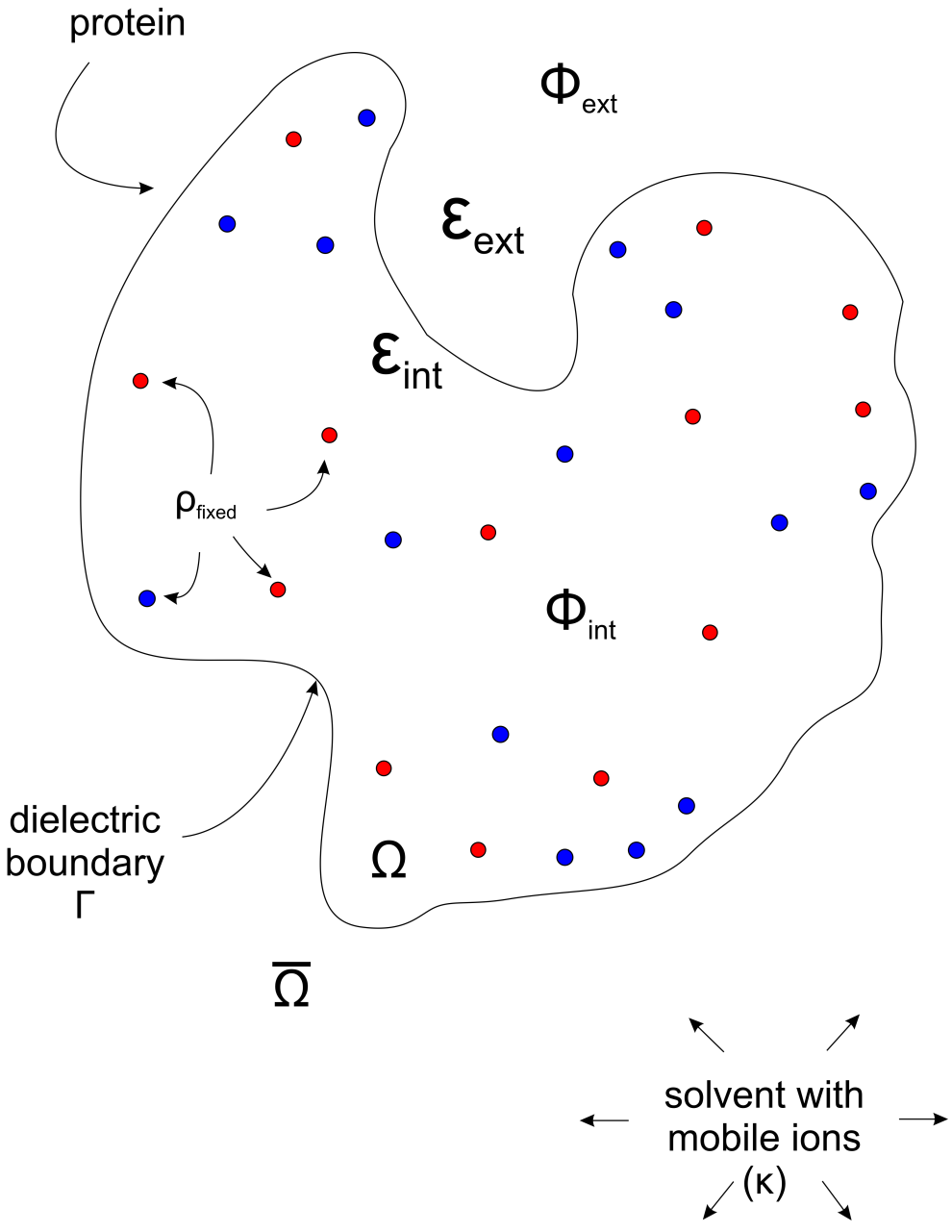


Figure 2.1: Continuum solvent model of a protein

equations in ϕ_{int} and ϕ_{ext}, as long as we can find a suitable scalar function to assign to ψ. The "trick" is to use the Green's functions for the partial differential equations in place of ψ, as follows. The Green's function for the Laplace equation is defined as the function $G(r_p, r_t)$ which satisfies the relation:

$$\nabla^2 G(r_p, r_t) = \delta(r_p - r_t) \tag{2.6}$$

where $\delta(r_p - r_t)$ is the Dirac delta function. (Note that this is a function of *two* variables representing positions, r_p and r_t.)

It is known that this Green's function (henceforth denoted G_{pt} for brevity) is:

$$G(r_p, r_t) = G_{pt} = \frac{1}{4\pi \left\| r_p - r_t \right\|} \tag{2.7}$$

Applying Green's 2nd Identity to the volume Ω with $\zeta = \phi_{int}(r_t)$ and $\psi = G_{pt}$ leads to:

$$\int_{\Omega} \left(\phi_{int}(r_t) \nabla^2 G_{pt} - G_{pt} \nabla^2 \phi_{int}(r_t) \right) d\Omega = \oint_{\Gamma} \left(\phi_{int}(r_t) \frac{\partial G_{pt}}{\partial n} - G_{pt} \frac{\partial \phi_{int}(r_t)}{dn} \right) d\Gamma_t \tag{2.8}$$

where all integrations are done over the variable position r_t which is within/on the boundary Γ of the volume Ω, whilst r_p (which appears implicitly in the Green's function G_{pt}) is held for the moment as some constant point *within* the volume[1].

Splitting the left hand volume integral into 2 parts and substituting the definition of the Green's function (2.6) we can write the first term as:

$$\int_{\Omega} \phi_{int}(r_t) \nabla^2 G_{pt} d\Omega = \int_{\Omega} \phi_{int}(r_t) \delta(r_p - r_t) d\Omega = \phi_{int}(r_p) \tag{2.9}$$

in which the volume integral of the delta function in effect returns the value of the function $\phi(r_t)$ at the point r_p.

In order to deal with the second part of the volume integral we substitute for $\nabla^2 \phi_{int}(r_t)$ using Equation 2.2 (noting that the position argument is r_t, whereas r_p is still some arbitrary constant point within the volume):

$$\int_{\Omega} - G_{pt} \nabla^2 \phi_{int}(r_t) d\Omega = \int_{\Omega} \frac{G_{pt}}{\varepsilon_{int}} \sum_k q_k \delta(r_t - r_k) d\Omega = \frac{1}{\varepsilon_{int}} \sum_k q_k G_{pk} \tag{2.10}$$

[1] We include a subscript t in our notation $d\Gamma_t$ to emphasize that the position r_t within the integral refers to the location of each surface increment $d\Gamma_t$, over which we are integrating.

where the delta function has "selected" the values of G_{pt} at the charge locations where $r_t = r_k$.

Combining Equations 2.9 and 2.10 into Equation 2.8 and expressing it all in terms of $\phi_{int}(r_p)$ gives a function for the internal potential as a function of the general position r_p within the volume.

$$\phi_{int}(r_p) = \oint_{\Gamma} \left(\phi_{int}(r_t) \frac{\partial G_{pt}}{\partial n} - G_{pt} \frac{\partial \phi_{int}(r_t)}{dn} \right) d\Gamma_t + \frac{1}{\varepsilon_{int}} \sum_k q_k G_{pk}, \qquad r_p \in \Omega \tag{2.11}$$

In order to follow a similar process for the external potential ϕ_{ext} we note that Equation 2.3 is the same as Poisson's Equation:

$$\left[\nabla^2 - \kappa^2\right] \phi_{ext} = 0 \tag{2.12}$$

for which it is known that the Green's function (which we shall denote u_{pt}) satisfying the definition:

$$\left[\nabla^2 - \kappa^2\right] u_{pt} = \delta(r_p - r_t) \tag{2.13}$$

is given by:

$$u_{pt} = \frac{exp(-\kappa \left\| r_p - r_t \right\|)}{4\pi \left\| r_p - r_t \right\|} \tag{2.14}$$

Thus applying Green's Second Identity to the external volume $\overline{\Omega}$ (i.e. that volume bounded by the molecular surface, as well as by an infinite surface at infinity[2]), and substituting $\zeta = \phi_{ext}(r_t)$ and $\psi = u_{pt}$, this time with r_p as an arbitrary constant point within the *external* volume and r_t being the integration variable:

$$\int_{\overline{\Omega}} \left(\phi_{ext}(r_t) \nabla^2 u_{pt} - u_{pt} \nabla^2 \phi_{ext}(r_t) \right) d\overline{\Omega} = \oint_{\Gamma} \left(\phi_{ext}(r_t) \frac{\partial u_{pt}}{\partial n} - u_{pt} \frac{\partial \phi_{ext}(r_t)}{dn} \right) d\Gamma_t \tag{2.15}$$

The definition of the Green's function (Equation 2.13) allows us to write the first part of the volume integral as:

[2]The surface integral turns out to be over the molecular boundary only since for the boundary at infinity ($r_t \to \infty$) the Green's functions tend to zero, as does the potential.

$$\int_{\overline{\Omega}} \phi_{ext}(r_t)\nabla^2 u_{pt} d\overline{\Omega} = \phi_{ext}(r_p) + \int_{\overline{\Omega}} \phi_{ext}(r_t)\kappa^2 u_{pt} d\overline{\Omega} \tag{2.16}$$

Equation 2.3 allows us to write the second part of the volume integral as:

$$\int_{\overline{\Omega}} - u_{pt}\nabla^2 \phi_{ext}(r_t) d\overline{\Omega} = - \int_{\overline{\Omega}} \phi_{ext}(r_t)\kappa^2 u_{pt} d\overline{\Omega} \tag{2.17}$$

Thus the sum of the two volume integrals convenient reduces to just $\phi_{ext}(r_p)$ giving:

$$\phi_{ext}(r_p) = \oint_{\Gamma} \left(\phi_{ext}(r_t) \frac{\partial u_{pt}}{\partial n} - u_{pt} \frac{\partial \phi_{ext}(r_t)}{dn} \right) d\Gamma_t \tag{2.18}$$

2.2.2 The Boundary Integral Equations as the point r_p approaches the boundary Γ

Equations 2.11 and 2.18 give the value of the potential at a points *not on* the boundary, in terms of surface integrals of the (scalar) values of the potential and its normal derivative *on* the boundary. The equations as written suggest a singularity at the boundary, since the Green's functions (2.7 and 2.14) will tend to infinite values as $r_p \to r_t$, and their derivatives are *strongly singular* (singularity of order r^{-2}).

The behaviour of the functions at the boundary can be resolved by considering a physical interpretation for the Green's function solutions. Considering the internal potential:

$$\phi_{int}(r_p) = \oint_{\Gamma} \left(\phi_{int}(r_t) \frac{\partial G_{pt}}{\partial n} - G_{pt} \frac{\partial \phi_{int}(r_t)}{dn} \right) dA + \frac{1}{\varepsilon_{int}} \sum_k q_k G_{pk}, \qquad r_p \in \Omega \tag{2.19}$$

this can be considered as the sum of a potential due to the fixed charge distribution ($\frac{1}{\varepsilon_{int}} \sum_k q_k G_{pk}$), plus the potential due to a "single layer" (i.e. a thin layer of charge smeared over the surface) given by $G_{pt} \frac{\partial \phi_{int}(r_t)}{dn}$ where $\frac{\partial \phi_{int}(r_t)}{dn}$ is the single-layer charge density, plus the potential due to a "double layer" (thin surface dipole layer) given by $\phi_{int}(r_t) \frac{\partial G_{pt}}{\partial n}$ where the dipole density is $\phi_{int}(r_t)$.

The Green's function solution has effectively represented everything outside of the volume as an equivalent combination of surface charge and dipole density over the surface. It is important to note that the single and double layer charge densities are not "real" in that the charges do not physically exist, but they are mathematically equivalent to the real potential distribution. This equivalence means that we can use the properties of single and double

layers (which are well known, see, e.g., Jackson [123] or Stratton [124]) to write the limits of 2.11 and 2.18 as the point r_p approaches the boundary [114]:

$$\phi_{int}(r_p) = \oint_{\Gamma}^{PV} \left[G_{pt} \frac{\partial \phi_{int}(r_t)}{\partial n} - \frac{\partial G_{pt}}{\partial n} \phi_{int}(r_t) \right] d\Gamma_t + \frac{1}{2}\phi_{int}(r_p) + \frac{1}{\varepsilon_{int}} \sum_k q_k G_{pk}, \qquad r_p \in \Gamma \tag{2.20}$$

$$\phi_{ext}(r_p) = \oint_{\Gamma}^{PV} \left[-u_{pt} \frac{\partial \phi_{ext}(r_t)}{\partial n} + \frac{\partial u_{pt}}{\partial n} \phi_{ext}(r_t) \right] d\Gamma_t + \frac{1}{2}\phi_{ext}(r_p), \qquad r_p \in \Gamma \tag{2.21}$$

where the label "PV" indicates the Cauchy principal value is required[3]. The effective single charge layer does not lead to any discontinuity in potential at the singular point, but the (infinitely thin) double layer produces a discontinuity which has a magnitude equal to the dipole density, that is to say $\Delta\phi = \phi(r_p)$. Taking the dielectric boundary to be a thin transition layer, over which the dielectric constant varies rapidly but continuously between the values ε_{int} and ε_{ext}, it can be shown that the potential across the boundary is also continuous [124]. Consequently the discontinuity implied by the double layer can be considered a "smooth" step function across the boundary, which is bisected by the boundary itself, leading to the limiting values of $\frac{1}{2}\phi(r_p)$ in Equations 2.20 and 2.21 [125].

Lu *et al.* [102, 104] note that the limit coefficient $\frac{1}{2}$ is strictly only valid for a planar surface[4], and that if the point r_p is the vertex of a polyhedron, Equations 2.20 and 2.21. should read:

$$\alpha_p \phi_{int}(r_p) = \oint_{\Gamma}^{PV} \left[G_{pt} \frac{\partial \phi_{int}(r_t)}{\partial n} - \frac{\partial G_{pt}}{\partial n} \phi_{int}(r_t) \right] d\Gamma_t + \frac{1}{\varepsilon_{int}} \sum_k q_k G_{pk}, \qquad r_p \in \Gamma \tag{2.22}$$

$$(1 - \alpha_p)\phi_{ext}(r_p) = \oint_{\Gamma}^{PV} \left[-u_{pt} \frac{\partial \phi_{ext}(r_t)}{\partial n} + \frac{\partial u_{pt}}{\partial n} \phi_{ext}(r_t) \right] d\Gamma_t, \qquad r_p \in \Gamma \tag{2.23}$$

where α_p is given by $\beta_p/4\pi$ and β_p is the solid angle at the vertex. Lu *et al.* claim that improved solutions for the surface potential are obtained when $\alpha_p = \beta_p/4\pi$, but do not carry

[3] i.e. the limit of the integrals as we approach the singularity from either side of the boundary.

[4] Depending on the discretization of the surface, it is usually argued that an arbitrarily small circular disc around the singular point approaches a plane, so the limiting value of $\frac{1}{2}$ is correct.

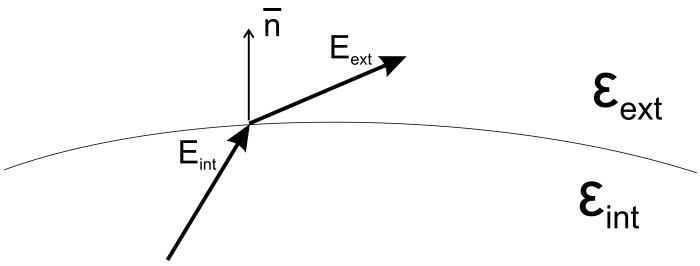


Figure 2.2: Boundary conditions

through the use of the geometric correction factor into their later work and choose to set $\alpha_p = \frac{1}{2}$.

So far we have two equations for four unknown sets of scalar values over the surface (ϕ_{ext}, ϕ_{int}, $\frac{\partial \phi_{int}}{\partial n}$, $\frac{\partial \phi_{ext}}{\partial n}$). However this can be reduced to two sets of unknown values by applying boundary conditions at the protein surface (Figure 2.2).

At the dielectric boundary, the potential (ϕ) is continuous, so for points $r_p \in \Gamma$ the condition $\phi_{int}(r_p) = \phi_{ext}(r_p)$ holds. Applying Gauss' Law to a small cylinder oriented across the boundary quickly demonstrates that the normal component of the electric displacement vector D on each side of the boundary must be equal (since there is no charge enclosed by the cylinder). Recalling that the electric displacement vector is related to the electric field E by the dielectric constant ε ($D = \varepsilon E$) allows us to write:

$$\varepsilon_{int} E_{int}(r_p) \cdot n_p = \varepsilon_{ext} E_{ext}(r_p) \cdot n_p \tag{2.24}$$

where n_p is the surface normal vector at point r_p. Note that since the potential is continuous over the boundary, the component of the electric field E parallel to the boundary must also be continuous: this means the electric field vector E will be "refracted" by the dielectric boundary as illustrated in Figure 2.2.

Noting that $E \cdot n = -\frac{\partial \phi}{\partial n}$, we can write this as:

$$\frac{\partial \phi_{int}(r_p)}{\partial n_0} = \varepsilon_{ratio} \frac{\partial \phi_{ext}(r_p)}{\partial n_0} \tag{2.25}$$

where ε_{ratio} is now the ratio of dielectric constants $\varepsilon_{ratio} = \frac{\varepsilon_{ext}}{\varepsilon_{int}}$. We also define the scalar functions:

$$f_p = \phi_{ext}(r_p) \tag{2.26}$$

$$h_p = \frac{\partial \phi_{ext}(r_p)}{\partial n_0} \tag{2.27}$$

for points $p \in \Gamma$ (where the normal vector n_0 is the outward-pointing normal at r_p). Using these boundary conditions and substitutions, we can write two equations for the 2 sets of unknowns (note we have divided through by ε in Equation 2.28):

$$\frac{1}{2\varepsilon} f_p = \oint_{\Gamma}^{PV} \left[-\frac{1}{\varepsilon_{ratio}} \frac{\partial G_{pt}}{\partial n} f_t + G_{pt} h_t \right] d\Gamma_t + \frac{1}{\varepsilon_{ext}} \sum_k q_k G_{pk}, \qquad p \in \Gamma \tag{2.28}$$

$$\frac{1}{2} f_p = \oint_{\Gamma}^{PV} \left[\frac{\partial u_{pt}}{\partial n} f_t - u_{pt} h_t \right] d\Gamma_t, \qquad p \in \Gamma \tag{2.29}$$

2.2.3 The derivative Boundary Integral Equations

In order to produce a numerically stable system of equations, it is necessary to combine linear combinations of the boundary integral equations with their derivatives [103, 114].

Differentiating Equations 2.20 and 2.21 with respect to the normal n_0 at the surface point r_p, then applying the same boundary conditions and variable substitutions as before, as well as dividing *both* equations by ε, we obtain:

$$\frac{1}{2} h_p = \oint_{\Gamma}^{PV} \left[-\frac{1}{\varepsilon_{ratio}} \frac{\partial^2 G_{pt}}{\partial n_0 \partial n} f_t + \frac{\partial G_{pt}}{\partial n_0} h_t \right] d\Gamma_t + \frac{1}{\varepsilon_{ext}} \sum_k q_k \frac{\partial G_{pk}}{\partial n_0}, \qquad p \in \Gamma \tag{2.30}$$

$$\frac{1}{2\varepsilon_{ratio}} h_p = \oint_{\Gamma}^{PV} \left[\frac{1}{\varepsilon_{ratio}} \frac{\partial^2 u_{pt}}{\partial n_0 \partial n} f_t - \frac{1}{\varepsilon_{ratio}} \frac{\partial u_{pt}}{\partial n_0} h_t \right] d\Gamma_t, \qquad p \in \Gamma \tag{2.31}$$

Combining the "normal" Equations 2.28 and 2.29 and the "derivative" Equations 2.30 and 2.31 gives us a new linear system of coupled equations which can be written:

$$\begin{aligned} \left(\frac{1}{2\varepsilon_{ratio}} + \frac{1}{2} \right) f_p \quad = \quad & \oint_{\Gamma}^{PV} \left[\left(G_{pt} - u_{pt} \right) h_t - \left(\frac{1}{\varepsilon_{ratio}} \frac{\partial G_{pt}}{\partial n} - \frac{\partial u_{pt}}{\partial n} \right) f_t \right] d\Gamma_t \\ & + \frac{1}{\varepsilon_{ext}} \sum_k q_k G_{pk}, \qquad p \in \Gamma \end{aligned} \tag{2.32}$$

$$\left(\frac{1}{2}+\frac{1}{2\varepsilon_{ratio}}\right)h_p = \overset{PV}{\oint_\Gamma}\left[\left(\frac{\partial G_{pt}}{\partial n_0}-\frac{1}{\varepsilon_{ratio}}\frac{\partial u_{pt}}{\partial n}\right)h_t-\frac{1}{\varepsilon_{ratio}}\left(\frac{\partial^2 G_{pt}}{\partial n_0 \partial n}-\frac{\partial^2 u_{pt}}{\partial n_0 \partial n}\right)f_t\right]d\Gamma_t + \frac{1}{\varepsilon_{ext}}\sum_k q_k \frac{\partial G_{pk}}{\partial n_0}, \qquad p\in\Gamma \tag{2.33}$$

These are the derivative BEM (dBEM) equations as written by Lu *et al.* [103] which were also derived somewhat earlier by Juffer *et al.* [114].

2.2.4 Multiple Molecules

Lu *et al.* [103] show that the derivation here for a single molecular boundary (i.e. one protein) can in fact be easily extended for multiple boundaries (proteins), as long as each boundary is a closed surface within the solvent. The details of the derivation are reproduced in the Appendix A (they are essentially the same process as used above in applying Green's second identity to the different volumes of protein). The final result is almost identical to Equations 2.32 and 2.33, merely with a summation over J molecular surfaces Γ^j:

$$\left(\frac{1}{2\varepsilon_{ratio}}+\frac{1}{2}\right)f_p = \sum_J \overset{PV}{\oint_{\Gamma^j}}\left[\left(G_{pt}-u_{pt}\right)h_t-\left(\frac{1}{\varepsilon_{ratio}}\frac{\partial G_{pt}}{\partial n}-\frac{\partial u_{pt}}{\partial n}\right)f_t\right]d\Gamma_t + \frac{1}{\varepsilon_{ext}}\sum_J\sum_{k^j} q_k G_{pk},\ p\in\Gamma^j,\ j=1,....J \tag{2.34}$$

$$\left(\frac{1}{2}+\frac{1}{2\varepsilon_{ratio}}\right)h_p = \sum_J \overset{PV}{\oint_{\Gamma^j}}\left[\left(\frac{\partial G_{pt}}{\partial n_0}-\frac{1}{\varepsilon_{ratio}}\frac{\partial u_{pt}}{\partial n}\right)h_t-\frac{1}{\varepsilon_{ratio}}\left(\frac{\partial^2 G_{pt}}{\partial n_0 \partial n}-\frac{\partial^2 u_{pt}}{\partial n_0 \partial n}\right)f_t\right]d\Gamma_t + \frac{1}{\varepsilon_{ext}}\sum_J\sum_{k^j} q_k \frac{\partial G_{pk}}{\partial n_0},\ p\in\Gamma^j,\ j=1,....J \tag{2.35}$$

2.2.5 Solving the BEM equations in matrix form

Assuming a surface discretization which gives a vector of N values for potential, $\mathbf{f}$, and its normal derivative $\mathbf{h}$, Equations 2.28 and 2.29 or Equations 2.32 and 2.33 can be re-arranged into a matrix-vector equation of the type $\mathbf{A}\bar{\mathbf{x}}=\bar{\mathbf{b}}$ where $\bar{\mathbf{b}}$ is a known vector of length $2N$, $\mathbf{A}$ is a square matrix of dimension $2N\times 2N$, and $\bar{\mathbf{x}}$ is the vector of unknown values, length $2N$.

Since it is known that the derivative BEM equations form a well-conditioned system of equations, we choose to use that formulation (again, following the derivation of Lu *et al.* [103]):

$$\begin{bmatrix} \left(\frac{1}{2\varepsilon_{ratio}} + \frac{1}{2}\right)\mathbf{I} + \mathbf{B} & -\mathbf{A} \\ \mathbf{D} & \left(\frac{1}{2\varepsilon_{ratio}} + \frac{1}{2}\right)\mathbf{I} - \mathbf{C} \end{bmatrix} \begin{pmatrix} \mathbf{f} \\ \mathbf{h} \end{pmatrix} = \frac{1}{\varepsilon_{ext}} \begin{pmatrix} \sum_{\mathbf{k}} \mathbf{q_k} \mathbf{G_{pk}} \\ \sum_{\mathbf{k}} \mathbf{q_k} \frac{\partial \mathbf{G_{pk}}}{\partial \mathbf{n_0}} \end{pmatrix} \tag{2.36}$$

where the right-hand-side bracketed term is a vector (total length $2N$) of Coulomb-like summations, over all[5] k charges (located at r_k, magnitude q_k), for each surface discretization point p. The term $\begin{pmatrix} \mathbf{f} \\ \mathbf{h} \end{pmatrix}$ is the set of unknowns, made from the concatenation of the $\mathbf{f}$ and $\mathbf{h}$ vectors (total length $2N$). The sub-matrices A, B, C, D in the left hand square brackets are each of dimension $N \times N$ and are formed from surface integral terms, A_{pt}, B_{pt}, C_{pt}, D_{pt}, (Equations 2.37-2.40), in which a surface integral is carried out over the discretized surface element Δt (part of the total surface, which is the union of all molecular surfaces: $\Gamma_1 \cup \Gamma_2 \cup \ldots \Gamma_J$); the point r_p corresponds to the point for the unknown value of $\mathbf{f}$ or $\mathbf{h}$ in that row of the matrix equation; ε_{ratio} is still the ratio of external to internal dielectric constants, ε_{ext} and ε_{int} respectively; $\mathbf{I}$ is the identity matrix.

$$A_{pt} = \int_{\triangle t} (G_{pt} - u_{pt})\, d\Gamma_t \tag{2.37}$$

$$B_{pt} = \int_{\triangle t} \left(\frac{1}{\varepsilon_{ratio}} \frac{\partial G_{pt}}{\partial n} - \frac{\partial u_{pt}}{\partial n} \right) d\Gamma_t \tag{2.38}$$

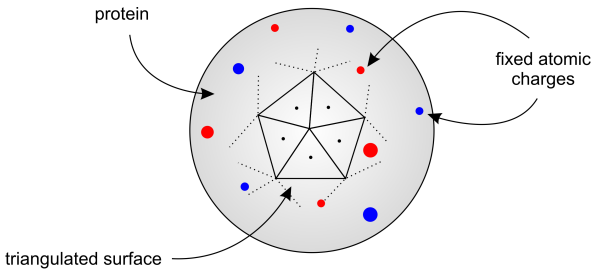
$$C_{pt} = \int_{\triangle t} \left(\frac{\partial G_{pt}}{\partial n_0} - \frac{1}{\varepsilon_{ratio}} \frac{\partial u_{pt}}{\partial n_0} \right) d\Gamma_t \tag{2.39}$$

$$D_{pt} = \int_{\triangle t} \frac{1}{\varepsilon_{ratio}} \left(\frac{\partial^2 G_{pt}}{\partial n_0 \partial n} - \frac{\partial^2 u_{pt}}{\partial n_0 \partial n} \right) d\Gamma_t \tag{2.40}$$

This matrix representation is illustrated in Figure 2.3 (diagonal terms $(\frac{1}{2} + \frac{1}{2\varepsilon})\mathbf{I}$ omitted to avoid complicating the Figure).

The surface integrations of the Green's functions over each element is non-trivial: care must be taken to avoid the singularities implied by the derivatives of the Green's functions when the point r_p is on or close to the element Δt.

[5] for multiple proteins, this summation is over *all* charges in all proteins



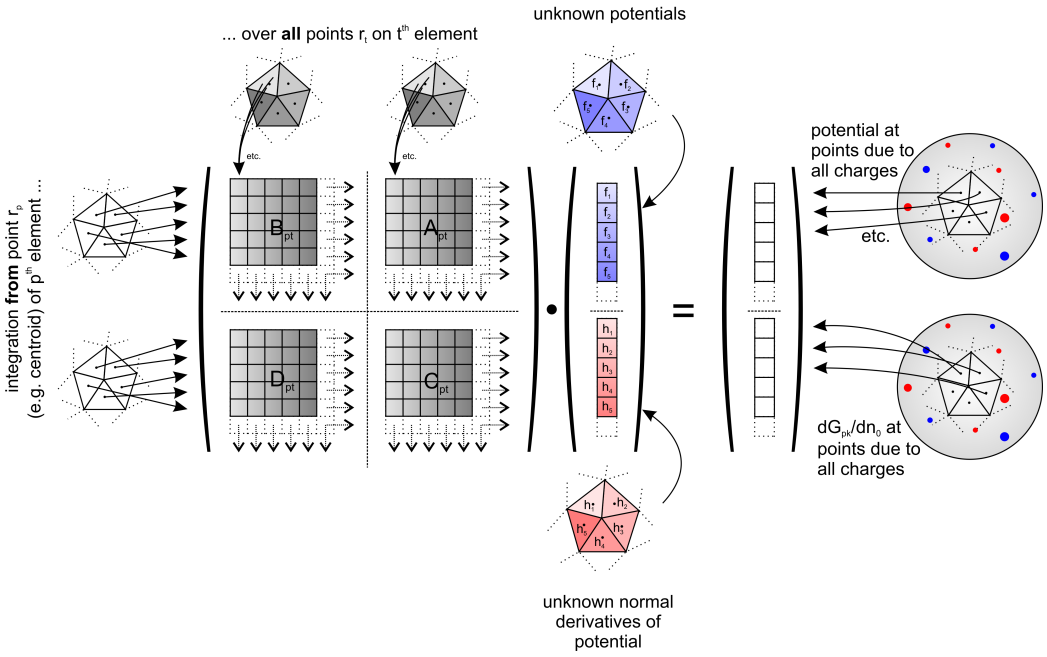


Figure 2.3: Illustration of the BEM matrix terms

In order to solve this system of equations where N is large, it is necessary to use an implicit "Krylov subspace" method such as GMRES [126], which solves the system of equations iteratively through successive evaluations of the matrix-vector product. The advantage of this method is that the matrix need not actually be present as a block of data in memory, as long as some function is available which returns the matrix-vector product; the disadvantage is if the matrix-vector product requires complicated numerical quadratures for each element of the matrix, then the computational requirement can become as burdensome as the memory overhead of storing the entire matrix!

The number of iterations required for a Krylov subspace method to converge on the correct solution depends on the condition number of the matrix, which in this case directly relates to how well conditioned the system of equations is.

At this point the BEM method appears to be $\mathcal{O}(N^2)$, since the matrix-vector product involves N^2 surface integrations: *from* each surface element *over* each element. Much work has been carried out in the last 10 years to accelerate the matrix-vector product by use of a far-field approximation such as the Fast Multipole Method (FMM) which is outlined in Section 2.3.

2.3 The Fast Multipole Method (FMM)

Given a set of N fixed charges in a weak ionic solution (Debye screening constant κ) which have magnitude q_i at points r_i, Coulomb's Law with an exponential screening factor can be used to calculate the net potential at any point r by the formula:

$$\phi(r) = \sum_{i=1}^{N} \frac{q_i e^{-\kappa \|r - r_i\|}}{4\pi\varepsilon_0\varepsilon_r \|r - r_i\|} \tag{2.41}$$

In order to solve the potential at each of the N points, the simplest method is to loop over each of the N charges, and carry out the above summation N times, yielding an $\mathcal{O}(N^2)$ algorithm.

The Fast Multipole Method (FMM) described in this section was developed by Greengard *et al.* [112, 127], and allows this to be achieved in $\mathcal{O}(N)$ operations, at the expense of accuracy: the FMM is an approximate method and guarantees accuracy only to within a specified error bound.

The FMM works by decomposing the volume using a tree structure (an octree) such that it is possible to collect charges in regions and create "multipole expansions" for those regions which approximate the net potential produced by those charges. These approximate functions are passed and aggregated throughout the tree in a hierarchical manner, such that

each region collects enough information to approximate the potential of the distant parts of the tree, but the total amount of work remains bounded.

The maximum error (i.e. the accuracy of the method) is controlled directly by the number of terms used in the multipole expansions.

The details are described in the following sections; these formulae are taken from the work of Huang *et al.* [113] which provides a convenient summary of the previous publications by Greengard *et al.* [112, 127, 128]. We begin by defining the multipole and local expansions, then introduce the octree, which gives a basis from which to explain the FMM algorithm; sections 2.3.5 to 2.3.8 fill in the details.

2.3.1 Notation

In order to avoid ambiguity the notation used in the following formulae mostly follows that of Huang *et al.* in [113]. Specifically, we assume that charges in general are located at Cartesian vector coordinates $\mathbf{x_i} = (x\mathbf{i} + y\mathbf{j} + z\mathbf{k})$ which can be expressed in spherical coordinates as $\mathbf{x_i} = (\rho_i, \alpha_i, \beta_i)$. An evaluation points for the potential is denoted $\mathbf{x}$ (no subscript) which has spherical coordinates $\mathbf{x} = (r, \theta, \psi)$. The constant factor $\frac{1}{4\pi\varepsilon_0\varepsilon_r}$ is omitted for simplicity. The main difference between the following formulae and those found in [113] is the choice of notation for the screening constant, which we choose to call κ (rather than λ) in order to maintain consistency with the notation of the Poisson-Boltzmann Equation.

2.3.2 Multipole Expansions

The potential at a distance due to a collection of charges within a sphere of radius a centred on the origin (given by the summation in Equation 2.41, and illustrated in Figure 2.4) can be written as a multipole expansion as follows (where the evaluation point $\mathbf{x}$ is located outside of the sphere, i.e. $r \geq a$):

$$\begin{aligned} \phi(\mathbf{x}) &= \sum_{i=1}^{N} q_i \frac{e^{-\kappa\|\mathbf{x}-\mathbf{x_i}\|}}{\|\mathbf{x}-\mathbf{x_i}\|} = \frac{2\kappa}{\pi}\sum_{i=1}^{N} q_i \cdot k_0\left(\kappa\|\mathbf{x}-\mathbf{x_i}\|\right) \\ &= \sum_{n=0}^{\infty}\sum_{m=-n}^{n} M_n^m k_n(\kappa r)\cdot Y_n^m(\theta,\psi) \end{aligned} \tag{2.42}$$

where the multipole coefficients M_n^m are given by:

$$M_n^m = 8\kappa\sum_{i=1}^{N} q_i \cdot i_n(\kappa\rho_i)\cdot Y_n^{-m}(\alpha_i,\beta_i) \tag{2.43}$$

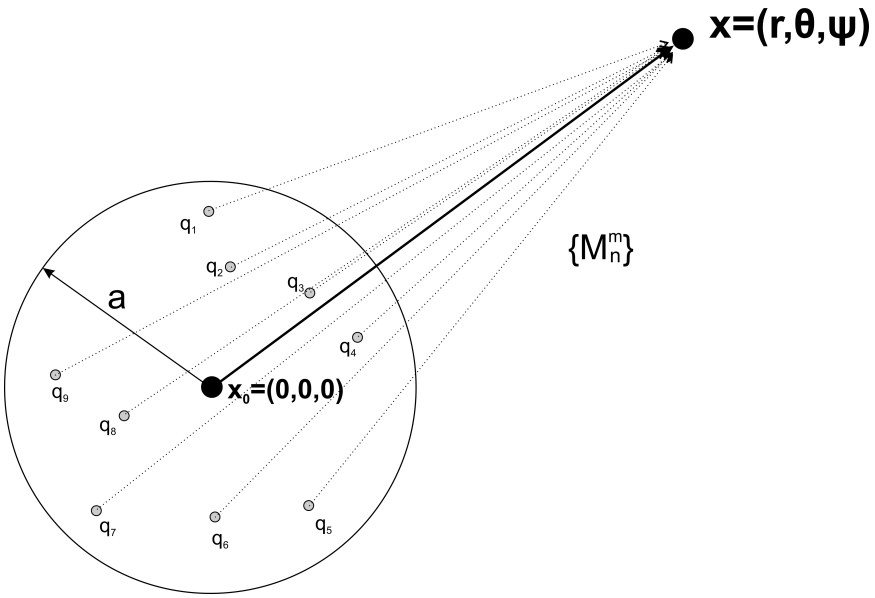


Figure 2.4: The multipole expansion: the potential arising from the set of charges q within the spherical region of radius a can be represented as a multipole expansion via the coefficients $\{M_n^m\}$. The expansion is valid for points outside the sphere, i.e. where $r > a$.

In the above formulae Y_n^m are spherical harmonic functions of degree n and order m, defined by:

$$Y_n^m(\theta,\psi) = \sqrt{\frac{2n+1}{4\pi}}\sqrt{\frac{(n-|m|)!}{(n+|m|)!}} \cdot P_n^{|m|}(cos\theta)e^{im\psi} \tag{2.44}$$

where P_n^m is the associated Legendre function, related to the Legendre polynomial $P_n(x)$ by:

$$P_n^m(x) = (-1)^m(1-x^2)^{\frac{m}{2}}\frac{d^m}{dx^m}P_n(x) \tag{2.45}$$

The functions $i_n(r)$ and $k_n(r)$ are the modified spherical Bessel and modified spherical Hankel functions respectively: they are related to the usual Bessel function $J_v(z)$ via various relationships which can be found in e.g. Abramowitz and Stegun [129]. For our purposes, it is enough to note that these functions are available directly from (freely available, open source) scientific library packages such as the GNU Scientific Library (GSL) [130] or within the special functions section of the Boost C++ libraries[6]. It is also worth noting at this point that these functions get very large or very small rapidly so in practice the multipole coefficients are scaled to avoid numerical overflow/underflow. These scaling coefficients are omitted from the mathematical treatment (as Greengard and Huang point out, the formulae

[6]at time of writing (2011), the Boost libraries can be found at `http://www.boost.org`. It is anticipated that the special functions in question will shortly be incorporated into the C++ standard (C++0x).

are complicated enough already [112]), but they must be used in the implementation of the algorithm.

Although the expansion looks somewhat complicated, in practice it is not too difficult to calculate these terms in a computer program: the result is a set of complex multipole coefficients M_n^m which encapsulate the effect of the set of charges. Given an evaluation point $\mathbf{x} = (r, \theta, \psi)$ we can return the potential by summing the multipole coefficients multiplied by the functions $k_n(\kappa r) \cdot Y_n^m(\theta, \psi)$.

It is not practicable to carry out the sum to infinite numbers of terms, therefore the expansion is truncated at a number of terms p:

$$\phi(\mathbf{x}) \approx \sum_{n=0}^{p} \sum_{m=-n}^{n} M_n^m k_n(\kappa r) \cdot Y_n^m(\theta, \psi) \tag{2.46}$$

The accuracy of the approximation scales according to the Equation 2.47 [112]: the accuracy depends on not only the number of terms (p) but also on the relative distance of the evaluation point (r) to the size of the region covered by the expansion (a).

$$\left| \phi(\mathbf{x}) - \sum_{n=0}^{p} \sum_{m=-n}^{n} M_n^m k_n(\kappa r) \cdot Y_n^m(\theta, \psi) \right| = \mathcal{O}\left(\frac{a}{r}\right)^p \tag{2.47}$$

According to Huang *et al.*, for an adaptive tree-based FMM, 21 terms is enough to yield 7 digit accuracy, or 42 terms for double precision accuracy [113].

2.3.3 Local Expansions

The truncated multipole expansion above approximates the potential due to a collection of charges within some bounding sphere (centred on the origin) at any remote point outside the sphere. A "local expansion" is a similar concept, representing the potential *within* some spherical region, due to a set of external charges which are located outside of the sphere, as illustrated in Figure 2.5.

Mathematically, this can be written as follows:

$$\phi(\mathbf{x}) \approx \sum_{n=0}^{p} \sum_{m=-n}^{n} L_n^m k_j(\kappa r) \cdot Y_n^m(\theta, \psi) \tag{2.48}$$

$$L_n^m = 8\kappa \sum_{l=1}^{N} q_l \cdot i_m(\kappa \rho_l) \cdot Y_n^{-m}(\alpha_l, \beta_l) \tag{2.49}$$

where $\phi(\mathbf{x})$ is the potential within the sphere, due to a set of N "far-field" charges, and L_n^m are local expansion coefficients. The number of terms p controls the accuracy of the

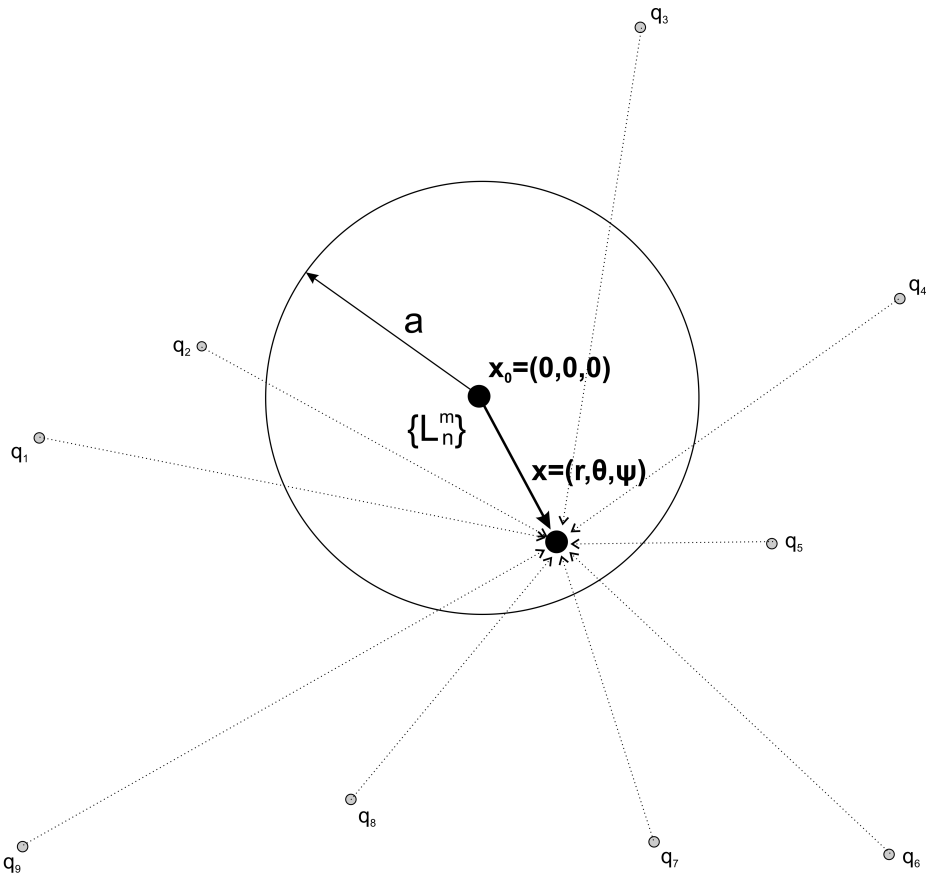


Figure 2.5: The local expansion: the local expansion coefficients $\{L_n^m\}$ represent the potential due to the exterior charges, valid for any point within the sphere of radius a.

approximation as for the multipole expansion (see Equation 2.47).

2.3.4 The octree based FMM

The FMM works by defining local expansions at evaluation points to calculate the potential due to "far-field" charges. The near-field must be added explicitly, since the local expansions cannot include the potential due to charges within the volume of the local expansion. Therefore, when it comes to evaluating the FMM there is always a set of explicit terms which scale like C^2 where C is the number of charges in the volume of the "near-field". The aim of the tree-code FMM is to limit the size of the local neighbourhood such that the explicit neighbourhood work per evaluation point (which is proportional to C^2) is a constant factor, independent of the total number of charges N.

Bearing this in mind, the octree is a hierarchical spatial decomposition: at the top level is a cube encompassing everything, which is recursively subdivided into 8 cubic "children" until no cube contains more than a certain number of charges, which we will denote by the constant C. Some important terms associated with the octree are defined as follows (and illustrated in Figure 2.6):

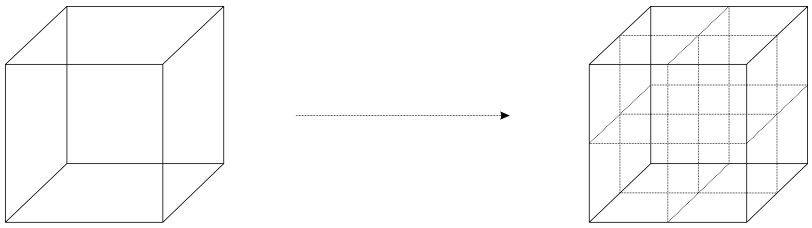
- Level - the level of the tree at which the cube resides (level 0 contains the root cube, it's children are on level 1, their children are level 2 etc.)
- Neighbourhood - the group of $3 \times 3 \times 3$ block of 27 cubes surrounding a cube of the tree[7], including the cube itself which is in the centre.
- Interaction List - the $6 \times 6 \times 6$ block of cubes which surround a cube of the tree, not including the neighbourhood. (N.B. the $6 \times 6 \times 6$ block of cubes correspond to the neighbourhood of the parent cube). Since the interaction list is $6 \times 6 \times 6$ with a $3 \times 3 \times 3$ block removed from the interior, the number of cubes in the interaction list of any given cube is: $216 - 27 = 189$[8].

We can now describe the fast multipole method in general terms.

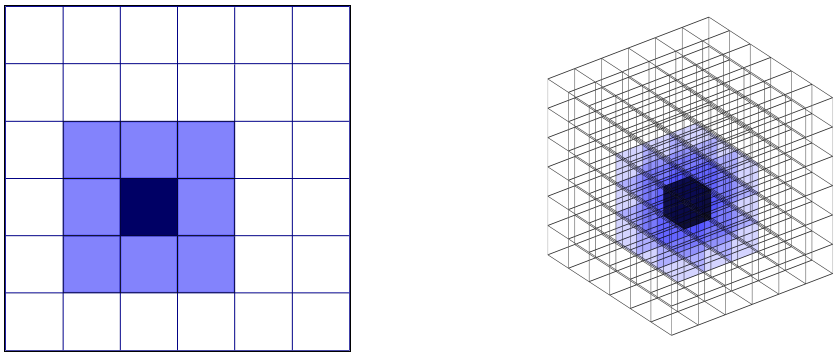
2.3.4.1 Upward pass

Multipole expansions for each cube are computed in an "upward pass" in which the bottom level (childless) cubes pass their multipole expansions up to the parent cube (not quite as trivial as it sounds: see Section 2.3.5), and so on up to the top of the octree.

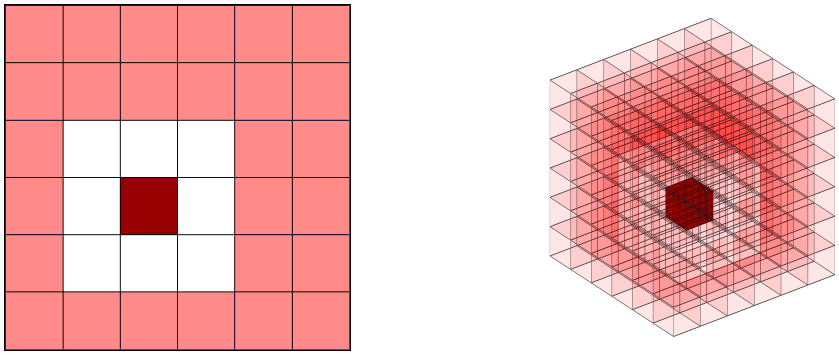
[7] Edge cubes will have fewer than 27 neighbours.
[8] The interaction list of edge cubes will contain fewer than 189 cubes.



(a) Parent to child subdivision of an octree cube



(b) The $3 \times 3 \times 3$ neighbourhood (light blue) of a cube (dark blue) (in 2D and in 3D)



(c) The interaction list (pink) of a central cube (dark red) in 2D and in 3D. (The neighbourhood in this case are the white squares/cubes.)

Figure 2.6: Spatial decomposition using an octree

Note that the charges are placed within cubic boundaries defined by the octree cubes, but the multipole expansions are valid only outside a *bounding sphere* of the cube, rather than the cube itself. Therefore the multipole expansions for a given cube X cannot be used to compute the potential in the immediate $3 \times 3 \times 3$ neighbourhood of X, because the bounding sphere spills into those cubes, as illustrated in Figure 2.7.

2.3.4.2 Downward pass

Following the upward pass each cube on every level has a multipole expansion. The downward pass transmits those expansions throughout the tree as follows.

Given a level of the tree, each cube X collects the multipole expansion from all remote cubes on that level which fulfill the following conditions:

1. the cubes are "well-separated" - i.e. the spherical boundaries of the two cubes do not overlap, so the multipole expansion of the remote cube is valid within cube X. In the octree this means the cubes are not in the $3 \times 3 \times 3$ neighbourhood of each other.

2. cube X has not already incorporated the multipole expansion of the remote cube, via inheritance from the parent level (see below).

The multipole expansions of remote cubes are translated through space and converted to local expansions through a somewhat complicated set of transformations, which are covered in Section 2.3.6. Taking this for granted for the time being, once the multipole expansions of remote cubes have been turned into a local expansion for cube X, cube X passes the local expansion down to its 8 children (if they exist). The children now have a local expansion which describes the potential due to everything in the tree *except* for the cubes which were omitted due to rule 1 above: i.e. the $3 \times 3 \times 3$ neighbourhood of X.

Once all the cubes on each level have completed this, the algorithm moves down to the next level of the tree.

The interaction list At this point we can observe that the interaction list for cube X (defined earlier as the $6 \times 6 \times 6$ region around that cube with a $3 \times 3 \times 3$ block removed) exactly corresponds to that region of cubes for which cube X did ***not*** receive multipole expansions via inheritance from the parent level, minus the immediate neighbourhood of X. The interaction list is the set of cubes which satisfy both conditions 1 and 2 above, hence the name "interaction list" meant to imply that it is the set of cubes for which multipole expansions must be translated and converted to local expansions. This is illustrated in 2 dimensions in Figure 2.8.

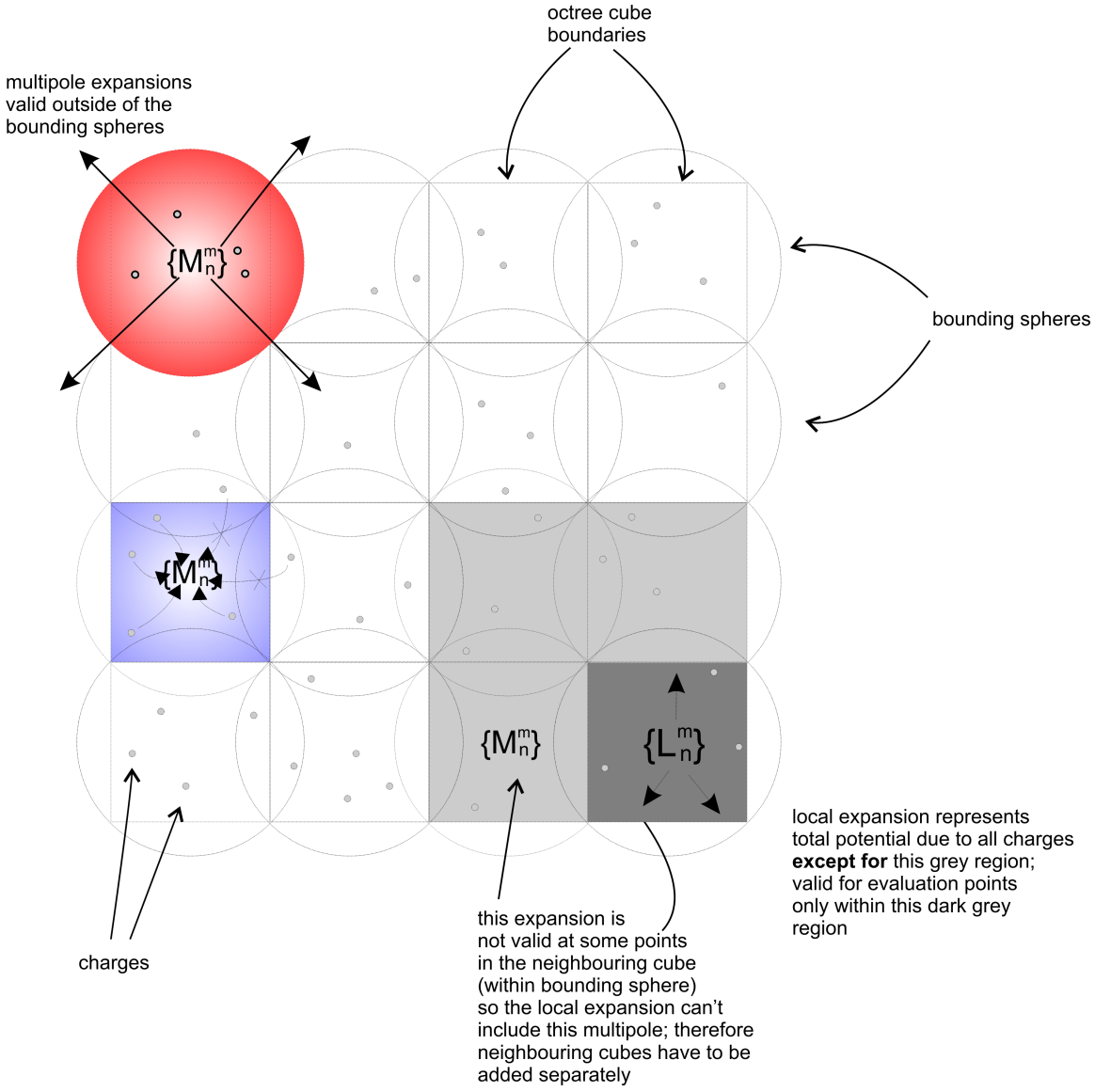


Figure 2.7: How multipole and local expansions represent the charges within a hierarchical tree

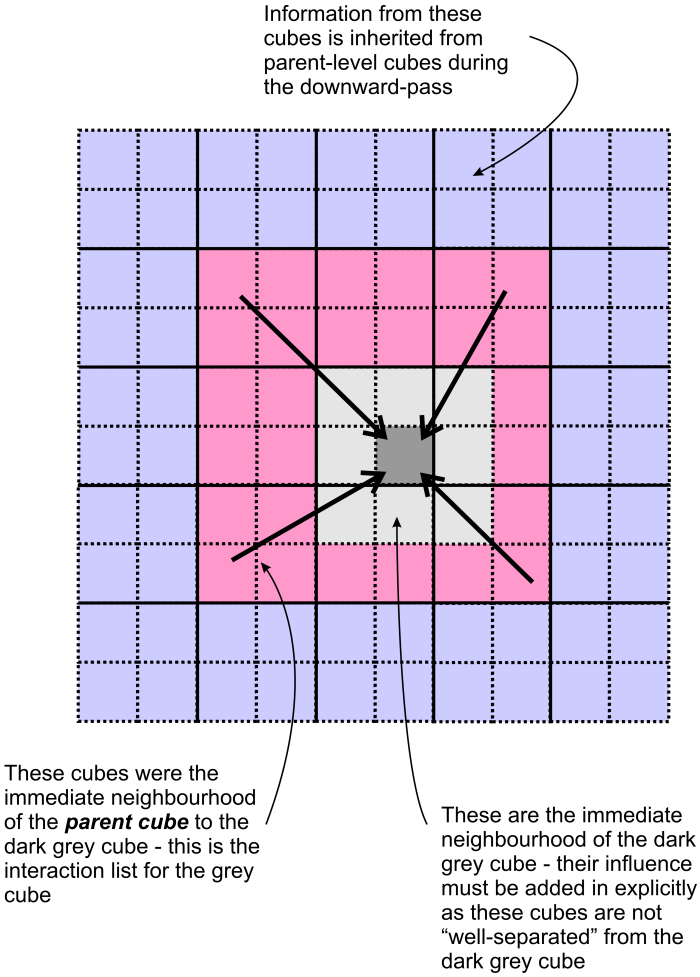


Figure 2.8: Building a local expansion for the right hand corner cell (dark grey) (in 2D): the blue region will already have been accounted for by the parent cube (local expansion obtained by the downward pass), so the effect of charges in that region are already taken care of; the light grey region is the immediate neighbourhood, multipole expansions cannot be taken from here as they are not "well separated"; this leaves the pink region (the interaction list), whose multipole expansions must be translated and converted to a local expansion.

2.3.4.3 Evaluation of local expansions and addition of explicit neighbour interactions

At the end of the downward pass, all bottom-level cubes should have a local expansion which approximates everything except the local neighbourhood. If the octree has been subdivided correctly, the maximum size of the neighbourhood is $27C$ (27 cubes, each with maximum C charges), so the maximum amount of work to explicitly evaluate the neighbourhood is $\propto 27C^2$.

2.3.5 Shifting Multipole/Local Expansions

This section and the following discuss in more detail the method by which the expansions are shifted/converted in the upward and downward passes.

The multipole/local expansions which represent the potential due to regions of charge can be combined by simple addition, as long as the coordinate frame of reference is the same for each expansion; the above definitions assume that the coordinate system has the origin at the centre of the octree cube, so expansions calculated for different cubes within the tree will not share the same origin. In order to combine, for example, 8 multipole expansions into a single multipole expansion centred on the parent octree cube the origins must be shifted to the new origin prior to adding the coefficients together.

We can define a linear operator T_{MM} for shifting multipole expansions, as illustrated in Figure 2.9. Using the definition for multipole expansions given previously (Equation 2.46), we can define $\{O_n^m\}$ as the multipole expansion to approximate the potential at $\mathbf{x}$ (outside of region D) due to the set of charges in the spherical region D (radius a, centred at $\mathbf{x_0} = (\rho, \alpha, \beta)$), as a function of the evaluation point expressed in terms of the relative position from the centre of D, $(\mathbf{x} - \mathbf{x_0}) = (r', \theta', \psi')$:

$$\phi(\mathbf{x}) \approx \sum_{n=0}^{p} \sum_{m=-n}^{n} O_n^m k_n(\kappa r') \cdot Y_n^m(\theta', \psi') \tag{2.50}$$

This can be re-written as an expansion in terms of position relative to the origin (i.e. into exactly the same form as Equation 2.46):

$$\phi(\mathbf{x}) \approx \sum_{n=0}^{p} \sum_{m=-n}^{n} M_n^m k_n(\kappa r) \cdot Y_n^m(\theta, \psi)$$

where now the region of validity is $r \geq (\rho + a)$. Although this looks like a simple change of variable, in fact the coefficients $\{O_n^m\}$ are distinct from $\{M_n^m\}$due to the change in region of validity for the expansion. The mapping between the two sets of multipole expansions is

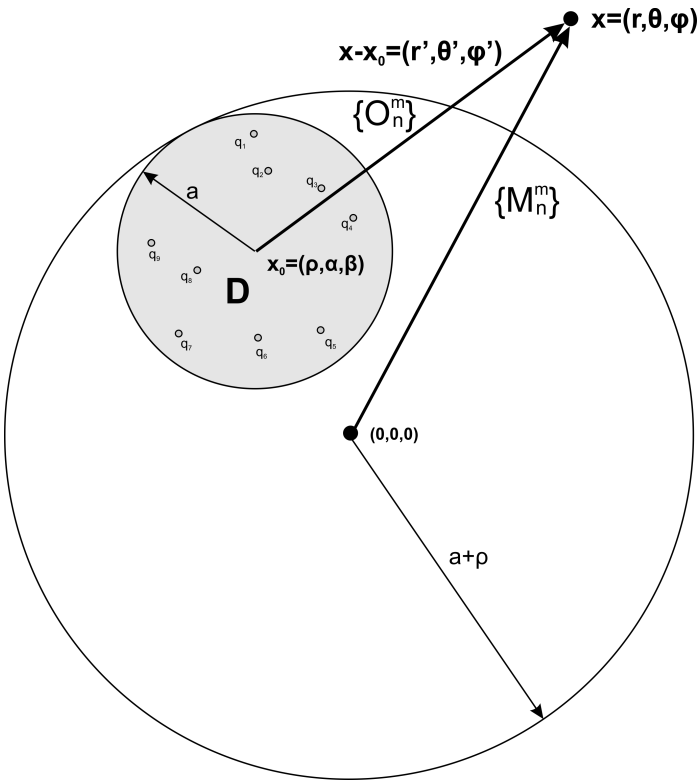


Figure 2.9: Translating a multipole expansion: the multipole expansion $\{O_n^m\}$ centred on x_0 represents the potential at any point outside of the shaded region (sphere of radius a), where the evaluation point is specified relative to x_0. In order to represent the same potential in terms of position relative to a different origin, the multipole expansion must be shifted to a new set of coefficients $\{M_n^m\}$. The potential evaluated via $\{M_n^m\}$ is now only valid outside of a larger spherical region of radius $a+\rho$. In order to combine several multipole expansions to aggregate the effects of distinct groups of charge, it is necessary to define a common origin for the aggregate expansion: each expansion must be shifted before adding the coefficients together in a linear fashion.

achieved by the linear operator T_{MM}; unfortunately the explicit formula for this is extremely complex [112] and its practical application would scale with p^4. Fortunately it is possible to devise a factored form of the translation operator, by noting that a shift purely in the z-direction is slightly easier to manage (see [112] for the details); if the expansion for region D in Figure 2.9 is centred on a point ρ units along the z-axis from the origin, then the shifted multipole coefficients can be expressed as:

$$M_n^m = \sum_{n'=m}^{p} C_m^{n,n'} \cdot O_n^m \tag{2.51}$$

where:

$$C_m^{n,n'} = \sum_{k=m}^{min(n,n')} \left(\frac{1}{2}\right)^k (-1)^{n'+k}(2n'+1)\frac{(n'-m)!(n+m)!(2k)!(\kappa\rho)^{-k} i_{n'+n-k}(\kappa\rho)}{(k+m)!(k-m)!(n'-k)!(n-k)!k!} \tag{2.52}$$

Although this expression for the translation looks complicated, the terms can be pre-computed and the computational cost of applying the translation to a set of multipole coefficients scales only with p^3. The caveat is that the translation is only valid for shifts in the z-direction, therefore for the general case in Figure 2.9 it is necessary to rotate the coordinate system of the multipole expansion coefficients O_n^m such that the z-direction within D lies parallel to the spherical angle (α, β) i.e. the z-direction points to the origin. This can be achieved by a rotation matrix $R(\alpha, \beta)$: the methods for rotating spherical harmonics using rotation matrices are covered in several textbooks [131]. Thus the mapping coefficient T_{MM} can be written as:

$$T_{MM} = R^{-1}(\alpha, \beta) \cdot T_{MM}^Z \cdot R(\alpha, \beta) \tag{2.53}$$

where T_{MM}^Z is a translation operator equivalent to T_{MM}, but rotated such that the translation is purely in the z-coordinate of the local coordinate frame.

A similar set of expressions exist for translating a local expansion (Figure 2.10): given a set of local expansion coefficients L_n^m which approximates the potential within a sphere (radius a, centred at the origin) due to charges outside of the sphere, the frame of reference can be shifted to a new origin within the sphere (N_n^m defined for points in the new coordinate frame $\mathbf{x} = (r', \theta', \psi')$) by converting the coefficients with a linear translation operator, T_{LL} which maps $\{L_n^m\}$to $\{N_n^m\}$. Once again, the explicit formula for T_{LL} is extremely complicated, however the diagonalised form for translations along the z-axis (T_{LL}^Z) uses the same

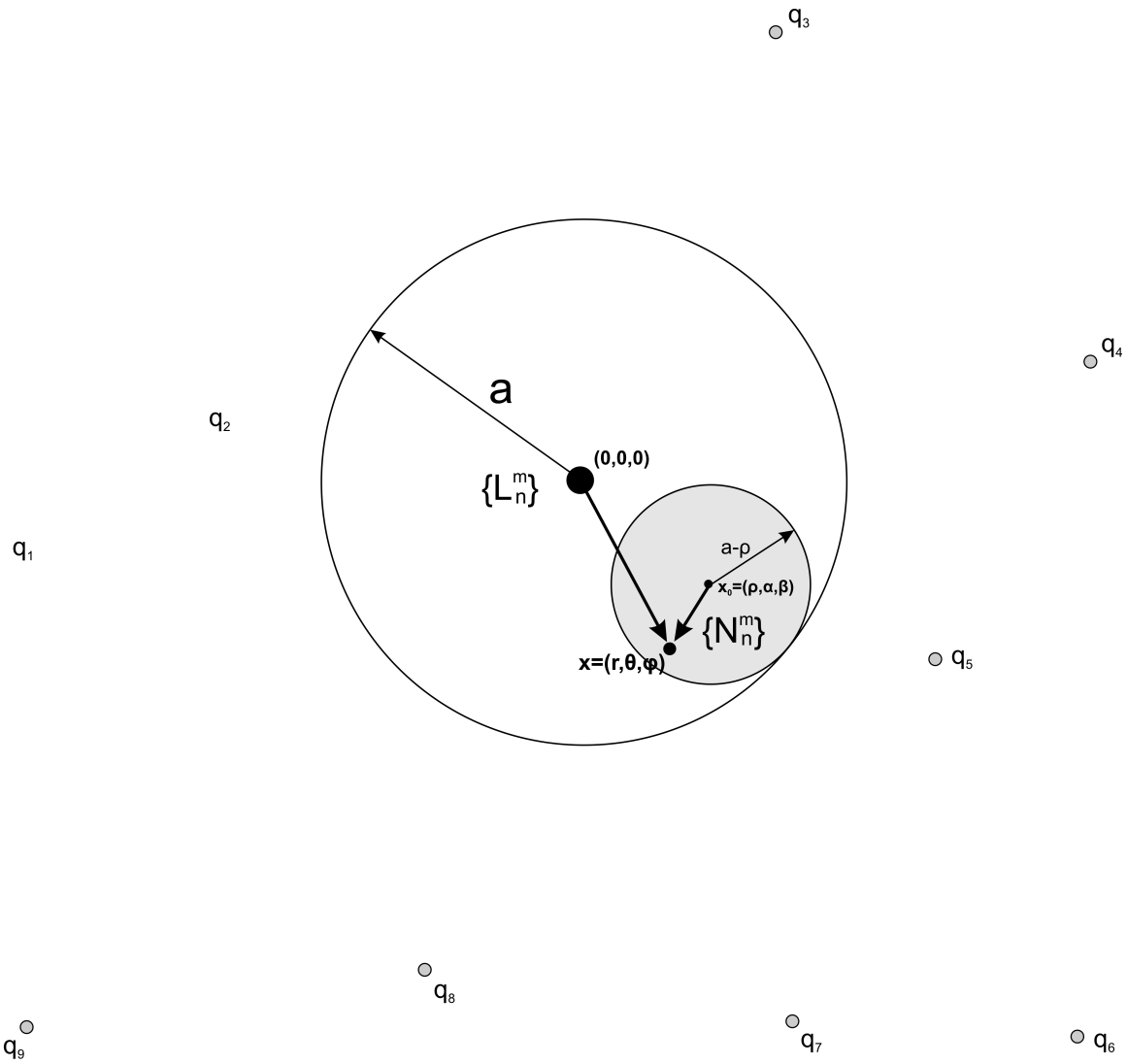


Figure 2.10: Translating a local expansion: the local expansion at the parent level$\{L_n^m\}$ centred on the origin $(0,0,0)$ represents the potential due to the external charges q. In order to express the local expansion in terms of location relative to a new origin, (e.g. that of the "child" region shaded grey) $x_0 = (\rho, \alpha, \beta)$, the local expansion coefficients must be shifted to $\{N_n^m\}$. The region of validity for the shifted expansion is now a smaller sphere of radius $(a - \rho)$ centred on x_0.

relationship as Equation 2.51, mapping old to new local expansions via coefficients $C_m^{n,n'}$ which are *almost* the same as those given above for multipole expansions:

$$C_m^{n,n'} = \sum_{k=m}^{min(n,n')} \left(\frac{1}{2}\right)^k (2n'+1) \frac{(n'-m)!(n+m)!(2k)!(\kappa\rho)^{-k} i_{n'+n-k}(\kappa\rho)}{(k+m)!(k-m)!(n'-k)!(n-k)!k!} \tag{2.54}$$

Thus the local translations are given by the transformation:

$$T_{LL} = R^{-1}(\alpha,\beta) \cdot T_{LL}^Z \cdot R(\alpha,\beta) \tag{2.55}$$

where a rotation of the coordinate system is necessary to allow the shift to be made purely in the z-direction (represented by T_{LL}^Z).

2.3.6 Converting Multipole to Local Expansions

In order to aggregate the multipoles of remote boxes during the downward pass it is necessary to collect the multipole expansions for all remote boxes into a single local expansion. It is possible to use the above shifting transformations to move the multipole expansions to the coordinate frame of the local box, however we have not yet explained how to convert them into a local expansion. It is possible to define a linear opeartor T_{ML} which carries out this transformation, however like the translation operators it is very complicated in explicit form [112] .

It turns out to be simpler to decompose the conversion into a set of equivalent steps. Greengard *et al.* introduced the use of a "plane wave expansion" [128] as a method for representing the multipole expansion in an alternative form which has particularly convenient translation operators. The advantage of this method is vastly improved scaling with respect to the number of multipole terms p. Although the entire FMM would still technically be $\mathcal{O}(N)$ without using the following procedure, the dominant scaling of $\mathcal{O}(p^4)$ would make it break even with $\mathcal{O}(N^2)$ methods only for comparatively large values of N (depending on the choice of p ($\rightarrow$accuracy) of course).

2.3.6.1 The plane wave expansion: multipole to plane-wave expansions

We begin the explanation of the plane wave expansion by noting that the modified spherical Hankel function of degree zero $k_0(\kappa r)$ (which appears in Equation 2.46 as the means by which a multipole expansion M_n^m is converted into a value for the potential[9]) can be written

[9] The modified spherical Hankel function of degree zero is closely related to the Green's function for the Poisson equation: $k_0(\kappa r) = \frac{\pi}{2}\frac{e^{-\kappa r}}{\kappa r}$.

as a double integral [112]:

$$k_0(\kappa r) = \frac{1}{4\kappa}\int_0^{\infty} e^{-(u+\kappa)(z-z_0)} \int_0^{2\pi} e^{i\sqrt{u^2+2u\kappa}\cdot((x-x_0)cos\alpha+(y-y_0)sin\alpha)} d\alpha du \tag{2.56}$$

where the function is evaluated relative to a source point $P = (x_0, y_0, z_0)$, and a target point $Q(x, y, z)$ with $z > z_0$ such that $r = \|P - Q\|$.

The inner integral in α can be evaluated by the trapezium rule, i.e. with the interval 0 to 2π split into χ equal parts; the number χ controls the level of detail of this integration, with increasing values of χ giving increasingly accurate approximations.

Yarvin and Rokhlin [132] derived a set of quadrature rules for integrating the outer integral in u, which when substituted into Equation 2.46 leads to the following equation for the potential at a point $\mathbf{x}$:

$$\phi(\mathbf{x}) \approx \kappa \sum_{k=1}^{s(\varepsilon)} \sum_{j=1}^{\chi_k} W(k,j) \cdot e^{-(u_k+\kappa)\cdot z} \cdot e^{i\sqrt{u_k^2+2u_k\kappa}\cdot((xcos(\alpha_{j,k})+ysin(\alpha_{j,k}))} \tag{2.57}$$

where (x, y, z) are the Cartesian coordinates of $\mathbf{x}$ and the coefficients $W(k, j)$ are given by:

$$W(k,j) = \frac{\pi w_k}{2\kappa\chi_k} \sum_{m=-p}^{p} i^{|m|} \cdot e^{im\alpha_{j,k}} \cdot \sum_{n=|m|}^{p} M_n^m \sqrt{\frac{2n+1}{4\pi}} \sqrt{\frac{(n-|m|)!}{(n+|m|)!}} P_n^{|m|}\left(\frac{\kappa+u_k}{\kappa}\right) \tag{2.58}$$

in which u_k and w_k are the Yarvin/Rokhlin quadrature points and weights; the angular terms $\alpha_{j,k}$ are given by $\frac{2\pi j}{\chi_k}$; χ_k is the number of points used for the trapezium rule at the k^{th} quadrature point; all subject to the condition given in Equation 2.59:

$$1 \leq z \leq 4,\ 0 \leq \sqrt{x^2+y^2} \leq 4\sqrt{2} \tag{2.59}$$

This condition corresponds to the geometric conditions required for the validity of the Yarvin-Rokhlin quadrature. It may also be noted that these conditions correspond to the geometric relationship between a cube and its interaction list.

This set of formulae represent the mapping of multipole coefficients M_n^m to plane wave (exponential) coefficients $W(k, j)$, and is denoted T_{ME}.

The total number of plane wave (exponential) terms, $S_{exp} = \sum_{k=1}^{s(\varepsilon)} \chi_k$ can be chosen such that the error bound of the potential as represented by the plane wave expansion is comparable to that inherent in the multipole expansion [128].

2.3.6.2 Translating plane wave expansions

One of the primary advantages of the plane wave expansion is that the translation operator T_{EE}^{z} in the z-direction is particularly simple to implement, being a multiplication of the existing plane wave expansion by terms in the form:

$$e^{-(u_k+\kappa)\cdot\Delta z} \cdot e^{i\sqrt{u_k^2+2u_k\kappa}\cdot((\Delta x\cdot cos(\alpha_{j,k})+\Delta y\cdot sin(\alpha_{j,k}))} \tag{2.60}$$

which can be pre-computed. Note that the translation must be in the z-direction and is subject to the conditions of Equation 2.59 (with $(\Delta x, \Delta y, \Delta z)$ in place of (x, y, z))

Once the multipole expansions are represented as plane waves, the shifting transformation is more efficient than using the diagonalized operators T_{MM}^{Z} and T_{LL}^{Z}. Within the FMM there would be a large number of applications of these operators (up to 189 per cube of the octree), so the cost of converting to plane waves is more than offset by the saving made in these translation operations.

2.3.6.3 Converting plane wave expansions to local expansions

Given the plane wave expansion of Equation 2.57 (i.e. approximation of the potential due to a set of charges) , an equivalent local expansion can be formed using Equation 2.61, which we denote as the transformation T_{EL}:

$$L_n^m = (-1)^n i^{|m|}\sqrt{4\pi}\sqrt{2n+1}\sqrt{\frac{(n-|m|)!}{(n+|m|)!}}\sum_{k=1}^{s(\varepsilon)} P_n^{|m|}\left(\frac{\kappa+u_k}{\kappa}\right) \times \sum_{j=1}^{\chi_k} W(k,j)\cdot e^{im\alpha_{j,k}} \tag{2.61}$$

2.3.6.4 Direction lists: the interaction list split into 6

The above procedure of shifting and converting multipole expansions requires rotations which depend on the geometric orientation between the two cubes. Within the interaction list, we can define 6 smaller lists which correspond to the rotations required to ensure plane wave translations meet the requirements of Equation 2.59. These sub-lists of the interaction list are termed the "direction lists": the Uplist, Downlist, Northlist, Southlist, Eastlist and Westlist. In this context, 'Up' refers to the positive z-direction, 'Down' is the negative z-direction. 'North' and 'South' are the postive and negative y-directions, respectively, and 'East' and 'West' the positive and negative x-directions.

Given any cube in the tree, the "Uplist" for that cube is the subset of the interaction list for which interaction list cubes are located more "Up" than North/South/East/West. Similarly the "Downlist" is those interaction list cubes which are located more units "Down"

than they are North/South/East/West of the central cube. The other lists are defined similarly. Cubes which are an equal number of units in each direction (e.g. 2 cubes Upwards, 2 cubes Northwards, and 2 Cubes Westwards of the central cube) are allocated to the list with the following precedence: $\Delta up/\Delta down > \Delta north/\Delta south > \Delta east/\Delta west$. Each cube of the interaction list belongs to exactly one of the direction lists. Figure 2.11 illustrates the division of the interaction list into the direction lists.

The interaction-list part of the downward pass, summarising all of the above, is illustrated in Figure 2.12.

2.3.7 Scaling of the FMM with number of charges

The total computational cost of the FMM depends on the number of boxes in the octree which we will denote B, and on the number of terms included in the multipole expansion, p. Assuming that the source charges are distributed fairly evenly, the number of boxes will be linearly dependent on the number of charges, N (given a maximum number of charges per box, C). If we assume that the number of boxes is large (i.e. the octree is not sparse) then each box in general has a neighbourhood of 27 boxes, and an interaction list of 189 (neglecting edge effects). Therefore the cost of the FMM algorithm is:

- Octree initialisation: $\mathcal{O}(B)$
- Upward pass: $\mathcal{O}(p^2)$ for each of B boxes
- Interaction lists: $6B$ conversions to/from plane waves $\approx 6B \times \mathcal{O}(p^4)$ + 189 translations $\approx 189B \times \mathcal{O}(p^2)$
- Downward pass: 8 $\mathcal{O}(p^2)$ operations for each of B boxes
- Evaluation of local expansions: $\mathcal{O}(p^2)$ for maximum of C charges per box
- Evaluation of explicit neighbourhood interactions: $O(C^2)$ operations

Since p and C are constants, and B is related (approximately) linearly to N, the total scaling with respect to problem size N is also linear.

2.3.8 Higher Order Derivatives

The FMM as presented solves the potential arising from a distribution of charges. It is possible to extract higher order spatial derivatives of the potential by using a well-known recursive relationship for derivatives of spherical harmonic functions [103] which is given in Appendix A.

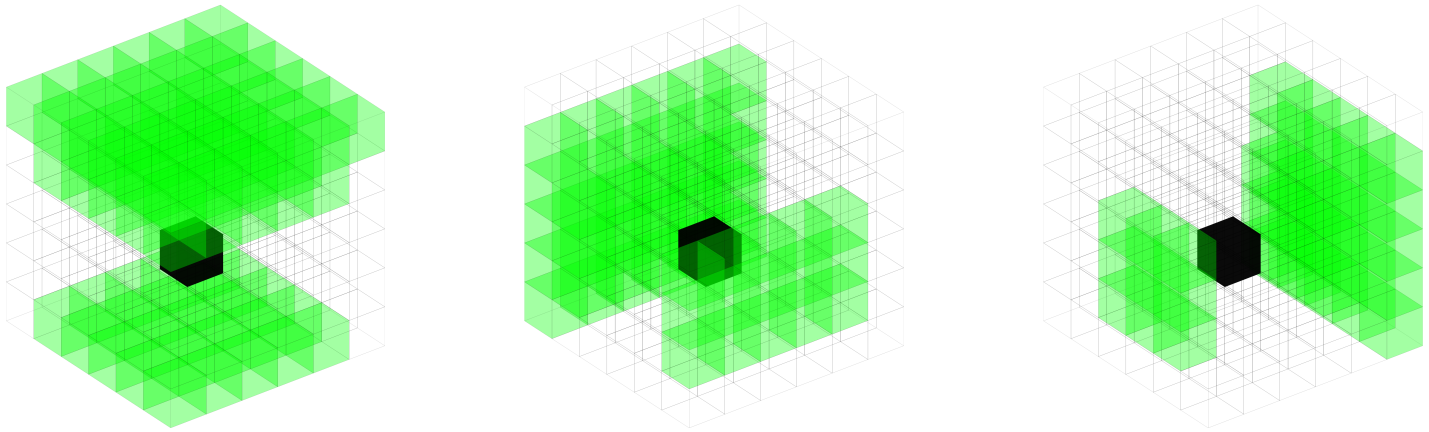
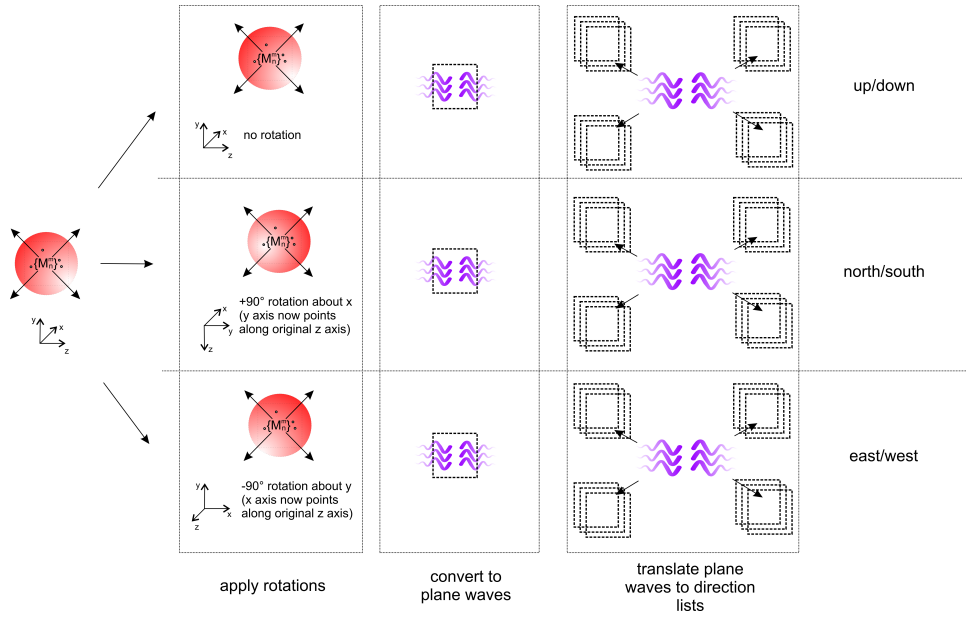
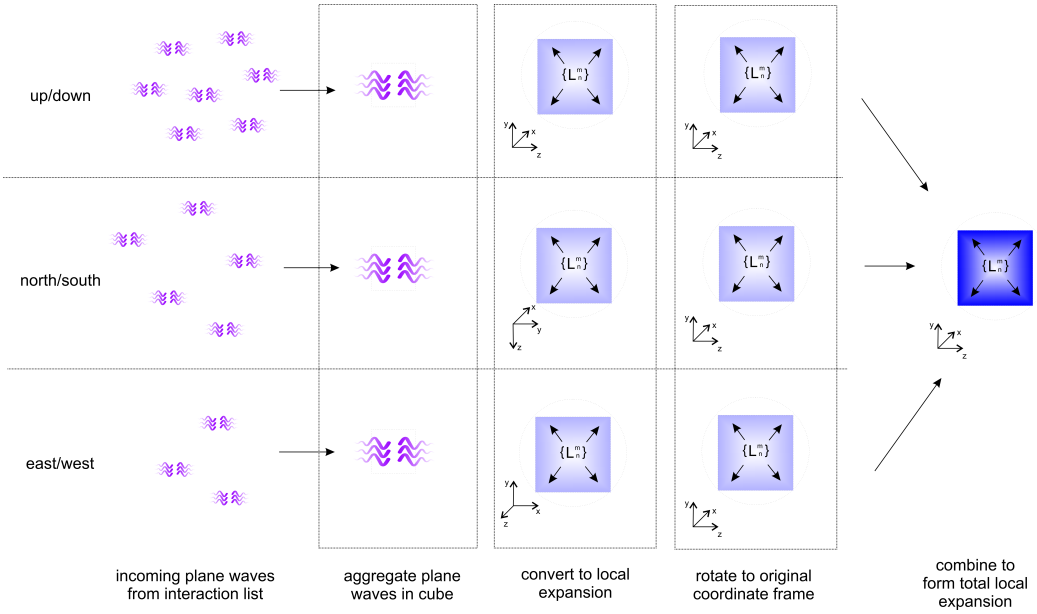


Figure 2.11: The direction lists for a cube in the octree (black). From left to right the green cubes represent: up/down; north/south; east/west lists.



(a) Multipole expansions for each cube of the FMM octree are converted to a set of plane-waves: one for each of the 6 directions (up/down, north/south, east/west). These plane waves are then translated to the remote interaction list nodes. (The plane-wave translations must be carried out in the local z-direction, so the multipole expansions are rotated appropriately prior to conversion.)



(b) Plane waves for each of the 6 directions are collected at each cube of the octree and then converted back to local expansions, which must be rotated back into the correct coordinate frame before being added together, producing a single local expansion for that cube.

Figure 2.12: The downward pass of the FMM: converting multipole expansions to local expansions, using plane waves

2.4 Combining the BEM with the FMM

In Section 2.2.5 we suggested that the $\mathcal{O}(N^2)$ BEM algorithm can be improved by using a far-field approximation within the matrix-vector product.

To explain how this can be achieved with the FMM described above, let us consider a single row in the upper half of the matrix in Equation 2.36. The approximation of the BEM integrals using the FMM is illustrated schematically in Figure 2.13.

The row-vector product for a row in the upper half requires a calculation of the surface integral over all surface elements, with respect to the point r_p, as defined by the sub-matrix kernels B_{pt} and A_{pt}. The results of each integral are multiplied by the corresponding value of f or h from the vector $\begin{pmatrix} \mathbf{f} \\ \mathbf{h} \end{pmatrix}$. The integrals B_{pt} and A_{pt} are equivalent to the calculation of a linear combination of potentials due to dipole density (B_{pt}), or charge density (A_{pt}); consequently, the multiplication of these surface integrals by the values of f and h respectively makes the row-vector product the equivalent of calculating the *total* "potential-like-function" at point r_p due to the "charge" represented by the values $\Delta S_t h_t$ (ΔS_t being the area of each element t) and the "dipoles" represented by $\Delta S_t f_t$. Expanding one of the row-vector products we can write:

$$\begin{aligned} \sum_t A_{pt} h_t &= \sum_t \left[(G_{pt} - u_{pt}) \Delta S_t h_t \right] \\ &= \sum_t \left[G_{pt} \left(\Delta S_t h_t \right) - u_{pt} \left(\Delta S_t h_t \right) \right] \end{aligned} \tag{2.62}$$

$$\begin{aligned} \sum_t B_{pt} f_t &= \sum_t \left[\left(\frac{1}{\varepsilon_{ratio}} \nabla G_{pt} + \nabla u_{pt} \right) \cdot \Delta S_t f_t n_t \right] \\ &= \sum_t \frac{1}{\varepsilon_{ratio}} \left[\begin{pmatrix} \frac{\partial G_{pt}}{\partial x} \\ \frac{\partial G_{pt}}{\partial y} \\ \frac{\partial G_{pt}}{\partial z} \end{pmatrix} \cdot \begin{pmatrix} \Delta S_t f_t n_x \\ \Delta S_t f_t n_y \\ \Delta S_t f_t n_z \end{pmatrix} - \begin{pmatrix} \frac{\partial u_{pt}}{\partial x} \\ \frac{\partial u_{pt}}{\partial y} \\ \frac{\partial u_{pt}}{\partial z} \end{pmatrix} \cdot \begin{pmatrix} \Delta S_t f_t n_x \\ \Delta S_t f_t n_y \\ \Delta S_t f_t n_z \end{pmatrix} \right] \end{aligned} \tag{2.63}$$

Written like this we can see that the right hand side of Equation 2.62 is $\frac{1}{\varepsilon_{ratio}}$ multiplied by the Coulomb potential (G_{pt}) at r_p arising from the set of "effective charges" of magnitude $\Delta S_t h_t$ at the positions of the elements Δ_t, minus the screened Coulomb potential (u_{pt}) arising from the same set of "effective charges". Equation 2.63 has a slightly more complicated interpretation: the scalar product terms on the right hand side represent the sum of Cartesian components of the electric field at r_p arising from the three different sets of "ef-

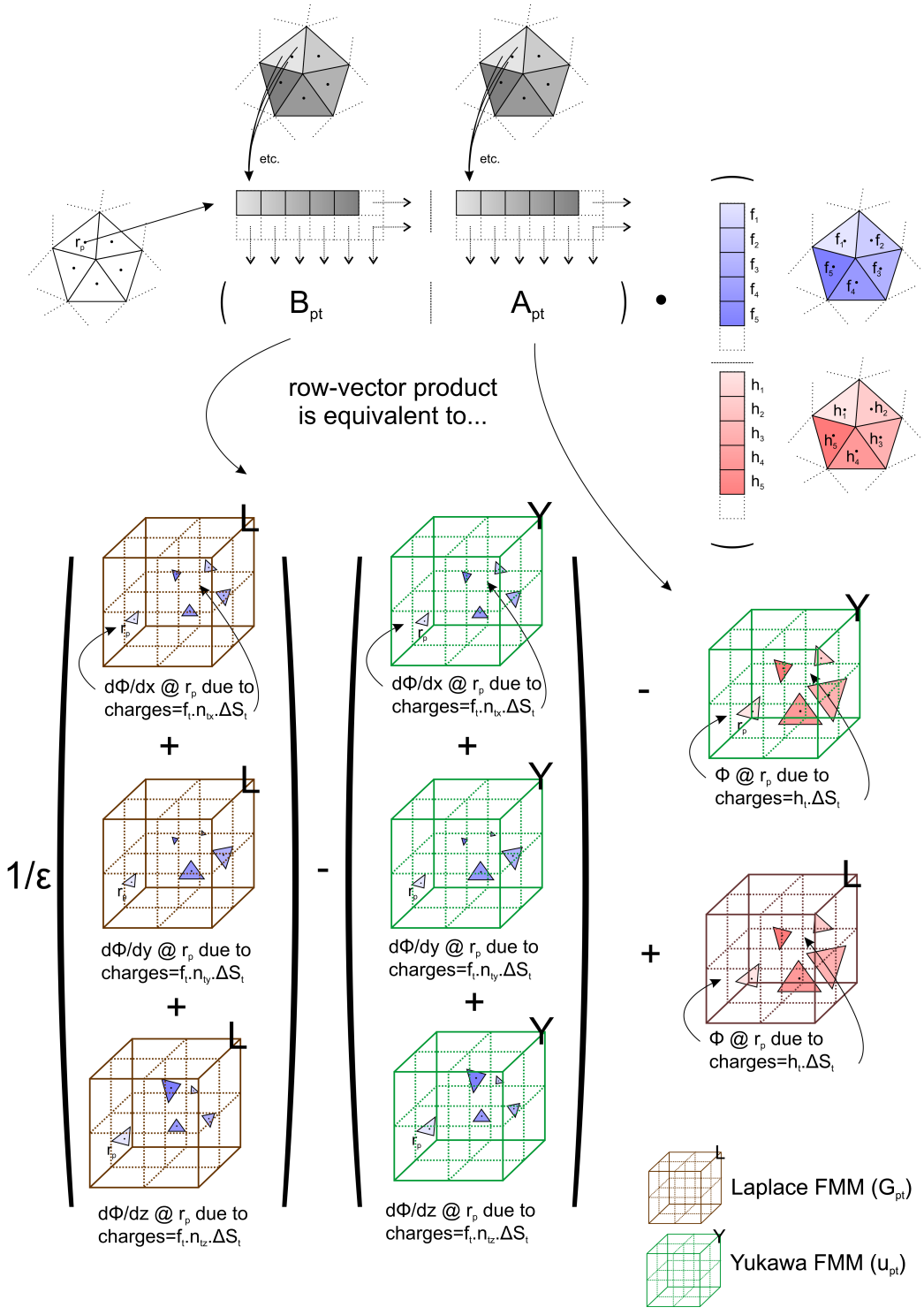


Figure 2.13: Approximating BEM matrix-vector terms with the FMM: the matrix row-vector products illustrated in Figure 2.3 are approximated by using several copies of the FMM to evaluate sets of effective potentials and higher derivatives which are equivalent to the BEM kernel functions A_{pt}, B_{pt}, C_{pt}, D_{pt}.

fective charge" quantities $\Delta S_t f_t n_x$, $\Delta S_t f_t n_y$, $\Delta S_t f_t n_z$, using either the unscreened Coulomb potential (∂G_{pt} terms) or screened Coulomb potential (∂u_{pt}).

In order to calculate the far-field A_{pt} and B_{pt} components, it is sufficient to apply the FMM to four different sets of effective charges: $\Delta S_t h_t$, $\Delta S_t f_t n_x$, $\Delta S_t f_t n_y$, $\Delta S_t f_t n_z$; two different FMM kernels are required: the screened Coulomb kernel for u_{pt} terms, and the unscreened Coulomb (or Laplace) kernel for G_{pt} terms[10]. Thus we must invoke the FMM 8 times (independently of one another) in order to obtain all of the terms needed to evaluate $A_{pt} h_t$ and $B_{pt} f_t$.

Similarly the effect of the row-vector product in the lower half of the matrix is the application of the kernels C_{pt} and D_{pt} on $\begin{pmatrix} \mathbf{f} \\ \mathbf{h} \end{pmatrix}$, which extracts a linear combination of "normal-derivatives-of-potential" due to the "charges" and "dipoles" $\Delta S_t h_t$and $\Delta S_t f_t$:

$$\begin{aligned} \sum_t C_{pt} h_t &= \sum_t \left[(\nabla G_{pt} \cdot n_0 - \frac{1}{\varepsilon_{ratio}} \nabla u_{pt} \cdot n_0) \cdot \Delta S_t h_t \right] \qquad (2.64) \\ &= \sum_t \left[\begin{pmatrix} \frac{\partial G_{pt}}{\partial x} \\ \frac{\partial G_{pt}}{\partial y} \\ \frac{\partial G_{pt}}{\partial z} \end{pmatrix} \cdot \begin{pmatrix} n_{0x} \\ n_{0y} \\ n_{0z} \end{pmatrix} \Delta S_t h_t - \frac{1}{\varepsilon_{ratio}} \begin{pmatrix} \frac{\partial u_{pt}}{\partial x} \\ \frac{\partial u_{pt}}{\partial y} \\ \frac{\partial u_{pt}}{\partial z} \end{pmatrix} \cdot \begin{pmatrix} n_{0x} \\ n_{0y} \\ n_{0z} \end{pmatrix} \Delta S_t h_t \right] \end{aligned}$$

$$\begin{aligned} \sum_t D_{pt} f_t &= \sum_t \left[\frac{1}{\varepsilon_{ratio}} \left(n_0 \cdot \nabla^2 G_{pt} \cdot n - n_0 \cdot \nabla^2 u_{pt} \cdot n \right) \right] \qquad (2.65) \\ &= \sum_t \frac{1}{\varepsilon_{ratio}} \begin{pmatrix} n_{0x} \\ n_{0y} \\ n_{0z} \end{pmatrix} \cdot \begin{pmatrix} \frac{\partial^2 G_{pt}}{\partial x \partial x} & \frac{\partial^2 G_{pt}}{\partial x \partial y} & \frac{\partial^2 G_{pt}}{\partial x \partial z} \\ \frac{\partial^2 G_{pt}}{\partial y \partial x} & \frac{\partial^2 G_{pt}}{\partial y \partial y} & \frac{\partial^2 G_{pt}}{\partial y \partial z} \\ \frac{\partial^2 G_{pt}}{\partial z \partial x} & \frac{\partial^2 G_{pt}}{\partial z \partial y} & \frac{\partial^2 G_{pt}}{\partial z \partial z} \end{pmatrix} \cdot \begin{pmatrix} \Delta S_t f_t n_x \\ \Delta S_t f_t n_y \\ \Delta S_t f_t n_z \end{pmatrix} \\ &\quad - \sum_t \frac{1}{\varepsilon_{ratio}} \begin{pmatrix} n_{0x} \\ n_{0y} \\ n_{0z} \end{pmatrix} \cdot \begin{pmatrix} \frac{\partial^2 u_{pt}}{\partial x \partial x} & \frac{\partial^2 u_{pt}}{\partial x \partial y} & \frac{\partial^2 u_{pt}}{\partial x \partial z} \\ \frac{\partial^2 u_{pt}}{\partial y \partial x} & \frac{\partial^2 u_{pt}}{\partial y \partial y} & \frac{\partial^2 u_{pt}}{\partial y \partial z} \\ \frac{\partial^2 u_{pt}}{\partial z \partial x} & \frac{\partial^2 u_{pt}}{\partial z \partial y} & \frac{\partial^2 u_{pt}}{\partial z \partial z} \end{pmatrix} \cdot \begin{pmatrix} \Delta S_t f_t n_x \\ \Delta S_t f_t n_y \\ \Delta S_t f_t n_z \end{pmatrix} \qquad (2.66) \end{aligned}$$

The row-vector product involving C_{pt} is interpretted in Equation 2.64 as the electric field $\begin{pmatrix} \frac{\partial G_{pt}}{\partial x} \\ \frac{\partial G_{pt}}{\partial y} \\ \frac{\partial G_{pt}}{\partial z} \end{pmatrix}$ evaluated at r_p arising from the set of effective charges $\Delta S_t h_t$, dotted with the

[10] In practice we can use the Yukawa (screened Coulomb) FMM for the G_{pt} terms as well by simply setting the screening parameter $\kappa = 0$, or more practically $\kappa = 10^{-10} \approx 0$ since the implementation of the FMM requires a non-zero value of κ for numerical stability.

normal vector at r_p, minus the same thing using the screened-Coulomb potential $\begin{pmatrix} \frac{\partial u_{pt}}{\partial x} \\ \frac{\partial u_{pt}}{\partial y} \\ \frac{\partial u_{pt}}{\partial z} \end{pmatrix}$ multiplied by $\frac{1}{\varepsilon_{ratio}}$. Within the algorithm this can be achieved through two applications of the FMM on the single set of charges $\Delta S_t h_t$ (i.e. once with the screened kernel, onece with the Laplace kernel). [11]

The D_{pt} components in Equation 2.65 are slightly more complicated. Here the three separate sets of charges used for the B_{pt} terms are again given the FMM treatment, this time with the 3×3 second derivatives matrix being evaluated at each point. The matrix-vector product:

$$\begin{pmatrix} \frac{\partial^2 G_{pt}}{\partial x \partial x} & \frac{\partial^2 G_{pt}}{\partial x \partial y} & \frac{\partial^2 G_{pt}}{\partial x \partial z} \\ \frac{\partial^2 G_{pt}}{\partial y \partial x} & \frac{\partial^2 G_{pt}}{\partial y \partial y} & \frac{\partial^2 G_{pt}}{\partial y \partial z} \\ \frac{\partial^2 G_{pt}}{\partial z \partial x} & \frac{\partial^2 G_{pt}}{\partial z \partial y} & \frac{\partial^2 G_{pt}}{\partial z \partial z} \end{pmatrix} \cdot \begin{pmatrix} \Delta S_t f_t n_x \\ \Delta S_t f_t n_y \\ \Delta S_t f_t n_z \end{pmatrix}$$

within Equation 2.65 can be expanded to a column vector (3 components x,y,z) with the x-component being composed of the sum of $\frac{\partial^2 G_{pt}}{\partial x \partial x}$ from the $\Delta S_t f_t n_x$ charge set, plus $\frac{\partial^2 G_{pt}}{\partial x \partial y}$ from the $\Delta S_t f_t n_y$ charge set, plus $\frac{\partial^2 G_{pt}}{\partial x \partial z}$ from the $\Delta S_t f_t n_z$ charge set; the other components are found in a similar fashion. The resultant vector undergoes a scalar product with the normal vector n_0 at r_p. The other matrix-vector term in Equation 2.65 represents the same process using the screened Coulomb potential kernel in the FMM rather than the Laplace kernel.

Finally we can see that the C_{pt} and D_{pt} components use the same set of effective charges as the A_{pt} and B_{pt} terms, so the 8 independent FMM operations already defined are sufficient, provided we evaluate the results to high enough derivatives of potential to account for all of the D_{pt} terms (up to second derivatives of potential are required).

At this point it is worth noting that the use of the FMM to evaluate the BEM integrals has made some automatic assumptions about the nature of the BEM solution: the effective charge sets inserted into the FMM are necessarily point charges, so the solutions ft and ht are in effect their average values over each element at the centre of mass of the element (or wherever the effective point charges are chosen to lie). The FMM in effect assumes that the G_{pt} and u_{pt} kernel functions are constant over the extent of the element which is equivalent to a crude 1-point numerical quadrature. This is likely to be an acceptable approximation as long as the evaluation point p is a long way from the far-field elements, and as long as

[11] Although C_{pt} looks similar in form to B_{pt}, it has a much simpler interpretation in the FMM because the derivative with respect to n_0 is at the evaluation point r_p, so in the FMM the "dot product" is carried out at the evaluation point; whereas the B_{pt} kernel has derivatives w.r.t the normal at each element, so the charges must be split into their normal components prior to the FMM.

the elements are not too large. Thus only the far-field part of the FMM should be used within the BEM; the near-field integrations must be done by some over form of numerical integration, with special care taken to avoid the singularities and near-singular functions when r_p and r_t are on the same/neighbouring surface elements.

As a final note the FMM is inherently an approximate method, so the accuracy of the approximation is also limited by the number of terms chosen within the multipole expansions.

2.4.1 Different dielectric constants for each macromolecule: 12 FMM evaluations instead of 8

In order to allow different macromolecules to adopt different dielectric constants the 8 independent FMM evaluations described above must be increased to 12 sets: 4 additional combinations comprise the effective charges of each patch scaled by the ratio of dielectric constants (ε_{ratio}) across that patch. Since the ratio of dielectric constant depends on which macromolecule the patch is a part of, the quantity ε_{ratio} which was constant in the above sections, becomes an attribute of each surface patch, $\varepsilon_{ratio,t}$.

The additional charge sets are:

$$\Delta S_t f_t n_x / \varepsilon_{ratio,t}$$
$$\Delta S_t f_t n_y / \varepsilon_{ratio,t}$$
$$\Delta S_t f_t n_z / \varepsilon_{ratio,t}$$
$$\Delta S_t h_t / \varepsilon_{ratio,t}$$

Once all 12 FMM sets are completed, they are recombined almost as before, according to the formulae Equations 2.62 to 2.65, except with the terms involving $\frac{1}{\varepsilon_{ratio,t}}$ replaced by the extra effective charge sets.

2.5 Summary

We have described in some detail the BEM as it applies to solving the linearized Poisson-Boltzmann Equation. In order to produce a method which scales linearly with the number of particles, it has been necessary to use a somewhat complicated variant of the FMM in order to linearize the matrix-vector product within the BEM formulation.

The general principles of how the BEM operates (as represented by solving the linear algebraic BEM matrix-vector equation), and an overview of the features of the FMM (structure of the octree hierarchically subdivision of space, the notions of upward and downward

passes) will be useful for the following chapters of this thesis where we describe our implementation of these algorithms. However the very fine mathematical detail (for example, how plane waves are obtained from multipole expansions) are not required to appreciate the remainder of the thesis.

Chapter 3

BEEP: Boundary Element Electrostatics Program

3.1 Outline of this chapter

We have implemented a program which we call BEEP (an acronym for Boundary Element Electrostatics Program) to solve the linearised PBE using the BEM/FMM algorithm. This chapter and associated appendices discuss the technical aspects of how the mathematical concepts of Chapter 2 are implemented in practice and shows that the program generates the correct results for idealized cases, for which analytic solutions are available for comparison. Although these test cases are much simpler than biological structures, the principles of software operation and the results of this chapter underpin the application of BEEP to the biological structures in Chapter 5.

3.2 How BEEP works

BEEP is designed to solve the linearised PBE for multiple proteins in solution, including the effects of salt, following the implicit solvation model for proteins outlined in Chapter 2. At its simplest, BEEP takes user input describing the molecular system, solves the linearised PBE, and returns the solution in terms of the electrical potential (f) and normal derivative of electric field (h) at the surface of the molecules. The exact details of the input vary a little depending on what the user is trying to achieve, but for the sake of argument, here we will assume that the user has a PDB structure of a protein and wants to know the solvation energy of that protein, in isolation, given certain values for the optional parameters which

in this case are the monovalent salt concentration (which we convert into an inverse Debye screening length, κ), internal dielectric constant ε_{int}, and external dielectric constant ε_{ext}. The work flow then resembles Figure 3.1.

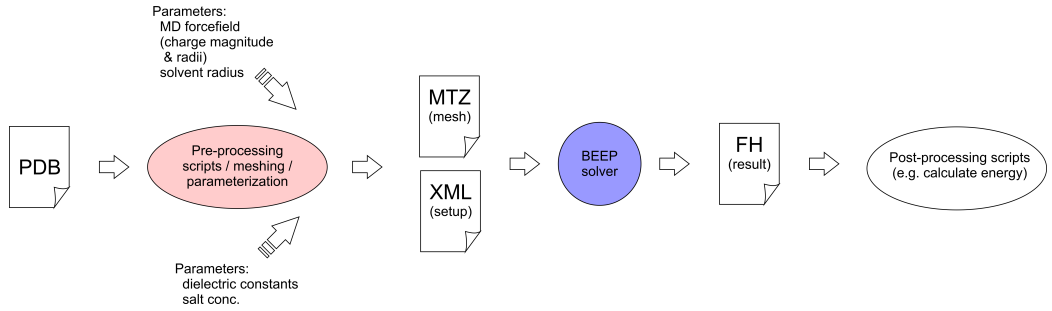


Figure 3.1: Diagram illustrating the basic work flow for finding the electrostatic solvation energy of a PDB protein structure.

The pre-processing step takes a PDB file and converts it to a form compatible with BEEP (by parameterising the atoms with partial charges and radii, and by building a surface mesh). The resultant "mtz" file can be passed into BEEP, along with coordinates and orientations in space of the protein, its dielectric constant, and the external solvent characteristics (i.e dielectric constant, salt concentration). BEEP then solves the linearised PBE for the system, and writes the output to a file. Post-processing scripts can be run on the output files to calculate quantities of interest, e.g. energies or forces.

More complicated arrangements of multiple molecules interacting can be built up by specifying combinations of meshed objects (i.e. mtz files) arranged in space as the input of BEEP, in which case BEEP will solve the system in exactly the same way but will produce multiple output files, each containing the surface solutions corresponding to one of the input mesh files. These can then be processed by the post-processing scripts as before.

This chapter begins with a brief description of how BEEP works, but for more in-depth details of input file formats for BEEP and a general discussion of practical issues, such as meshing, please refer to Appendix B.

3.2.1 BEEP internal working: solving the linearised PBE using BEM/FMM

The general procedure should be familiar from the descriptions of Chapter 2, but a short summary of the key ideas from that chapter are illustrated in Figure 3.2 for convenience. Given the input files described in the previous section, BEEP solves the system according to the work flow illustrated in Figure 3.3.

The most time consuming part of algorithm is the computation of the matrix-vector product, which is carried out using the FMM for far-field integrations, and explicit numerical

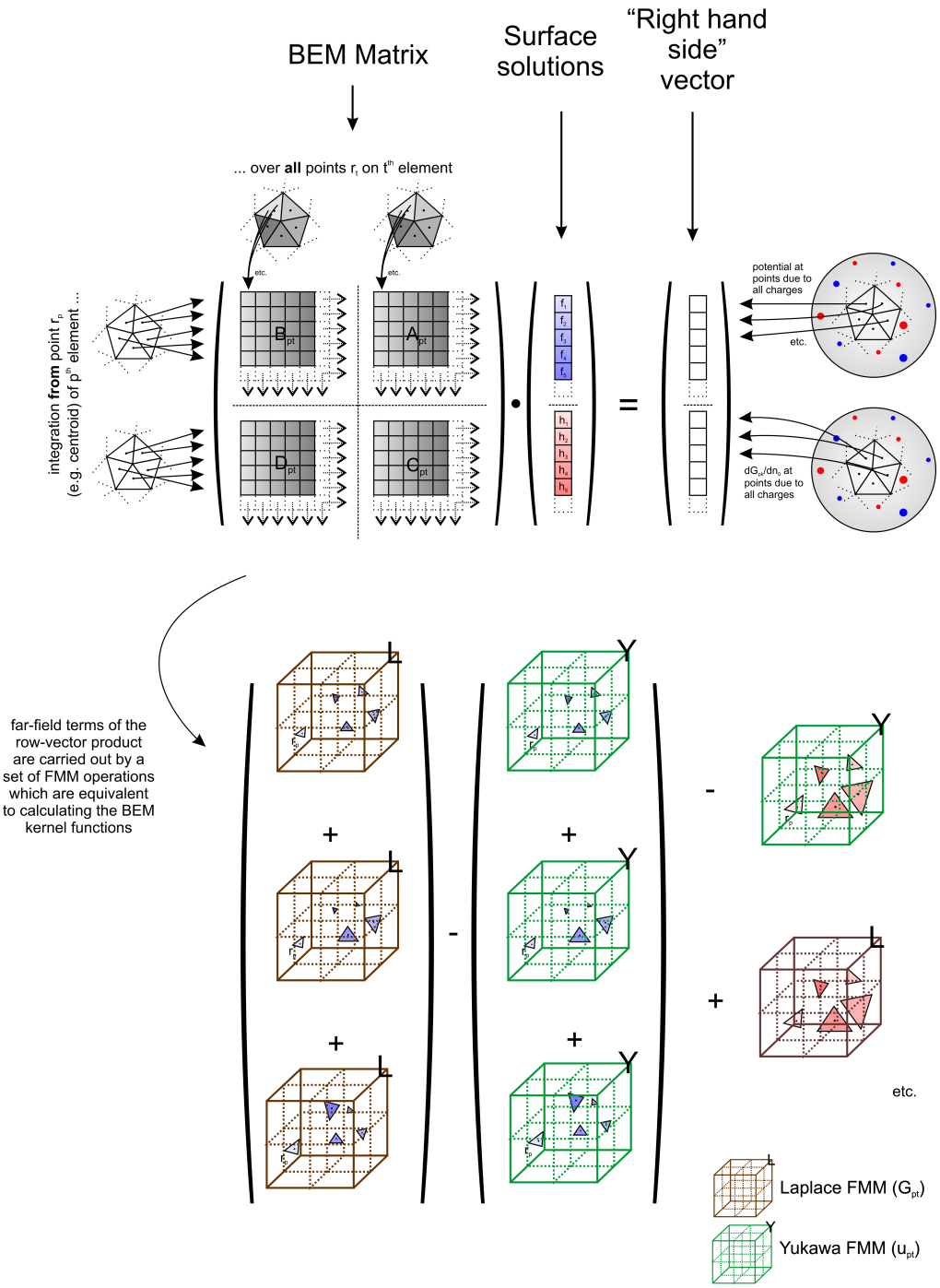


Figure 3.2: Solving the linearised PBE using BEM/FMM. This illustration summarises the key ideas from the previous chapter: the discretization of the BEM equations leads to a matrix-vector equation, the solution of which is the set of surface potentials and normal electric field components. The far-field terms in each row of the the matrix-vector multiplication are handled by a set of FMM evaluations, resulting in an algorithm that scales linearly with the number of surface points.

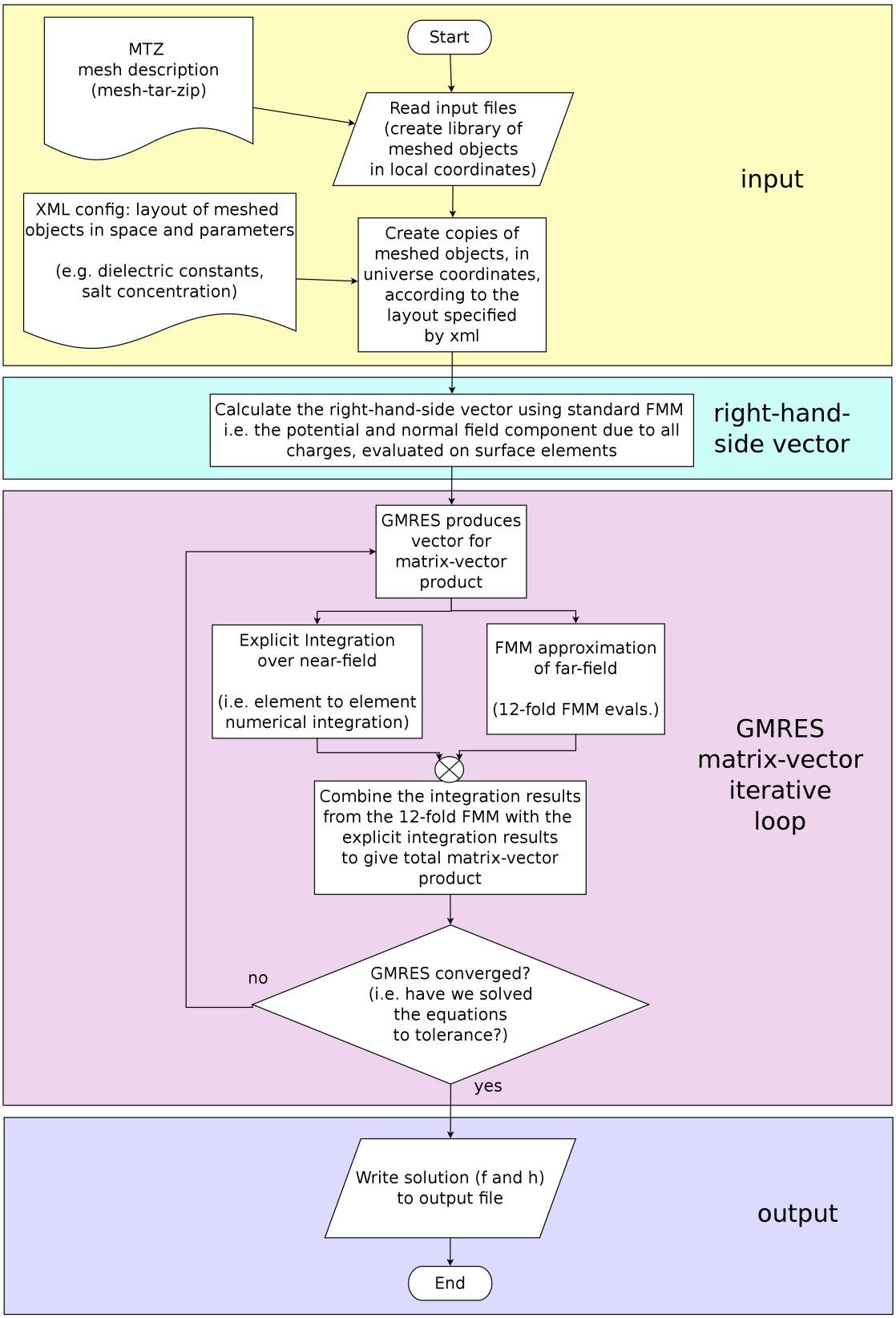


Figure 3.3: Flow diagram illustrating how BEEP solves the linearised PBE

integrations for the near-field. We ported a version of the FMM described in Chapter 2 from Fortran (supplied by Jingfang Huang [112]) to C++ (for various reasons, including performance considerations, but mostly to enable us to parallelise the algorithm more easily, as described in Chapter 4). The explicit integrations were carried out by the methods described in the following section.

3.2.2 Numerical Integration in BEEP

In the previous chapter we introduced the boundary element method as a way to solve the boundary integral equations by carrying out integrations over a "discretized surface". We shall now clarify what exactly we mean by this, as these details are crucial to the validity of BEEP results (and biomolecular BEM in general). The most important point to bear in mind is that we are aiming to convert from a system of continuous integral equations (Equations 2.34 and 2.35), defined in terms of an infinite number of infinitely small surface points p) into a discrete linear algebra matrix-vector equation (Equation 2.36, defined in terms of a finite number of surface elements Γ_t). The unknown continuous functions f and h become discrete values defined at specific points on the surface, which we must find by solving the linear system of equations.

3.2.2.1 Meshing, shape functions and basis functions

The task of creating a surface mesh to describe the dielectric boundary has been well addressed in the past, and numerous software packages[1] are available which define a molecular surface for a collection of spherical charges, and then decompose that surface into elements, e.g. planar triangles or quadrilaterals.

The choice of element shape (or more generally of shape functions) and the number of elements used controls the extent to which the surface mesh matches the "true" surface. The element shapes are normally chosen to be simple for obvious reasons: ease of mesh generation (without holes and without massive computational effort); compact storage of the surface elements; simplicity of programming algorithms to manipulate the surface elements; existence of algorithms to numerically integrate arbitrary functions over the surface elements. The simplest element shape is the planar triangle.

[1] e.g. MSMS, Meshlab, GAMER: see Appendix B for more details. In practice the task of obtaining a surface mesh with "high quality" triangles without defects (such as holes, self-intersecting faces, isolated triangles) is non-trivial, even using the advanced mesh generation tools in Meshlab, and a certain amount of manual intervention is necessary. By "high quality" we mean triangles which are non-slender such that their areas are non-tiny: very slender triangles tend to lead to unstable numerical integration in the BEM as they will result in a bias towards integrating the near-singular parts of the BEM kernel functions.

Once we have a surface mesh we require a way to map a continuous variable onto mesh elements as a function of position: this is the role of basis functions. Commonly the continuous variable is assumed to adopt a constant value across each element, or alternatively a linear function of position within the element (e.g. through linear interpolation of the vertex values). The discrete values of the continuous variable can therefore map to either values at the centroids of the elements or at their vertices. In general, other possibilities exist (such as defining a combination of spherical harmonic functions to approximate a variable on a spherical surface [115]), but these are not commonly found in BEM protein electrostatics literature.

BEEP is based upon planar triangle elements with a constant basis function, with a clever modification introduced by Lu *et al.* which they call the "node-patch".

3.2.2.2 The node-patch

Based on an initial mesh of planar triangles, Lu *et al.* [102] describe a "node-patch" scheme whereby the solution values of f and h are defined for each vertex of the surface. Each node-patch is defined as an "umbrella shaped" area around each vertex, created by dividing the initial triangles into six smaller sub-triangles, as illustrated in Figure 3.4 and Figure 3.5. The node-patch element shape is almost as easy to handle computationally as planar triangles, since each node-patch is simply a collection of planar sub-triangles: each triangle on the original mesh which shares that vertex contributes two sub-triangles to the node-patch, so a vertex on the surface which is shared by six triangles becomes a node-patch composed of twelve sub-triangles.

Lu *et al.* have shown that the use of a composite element shape of this type gave improved "accuracy"[2] over simple planar triangles, and even comparable accuracy to a linear basis function over planar triangles [99]. Apart from apparently improved accuracy the method reduces the number of unknowns since there are always many fewer vertices on a triangulated mesh than elements. This is possibly surprising because each solution point (vertex of a node-patch) must represent a greater area than that obtained when using the original planar triangle elements, so one would intuitively expect the overall error in the solution to be greater. We will suggest some reasons for this in Section 3.2.2.8, once we have introduced a few more relevant concepts. In Chapter 5 we will show how the node-patch can be improved even further by basing the mesh on curved Bézier-triangle elements rather than planar triangles.

[2] By accuracy we mean with regard to the calculation of solvation energy of simple spherical systems for which an analytic solution exists

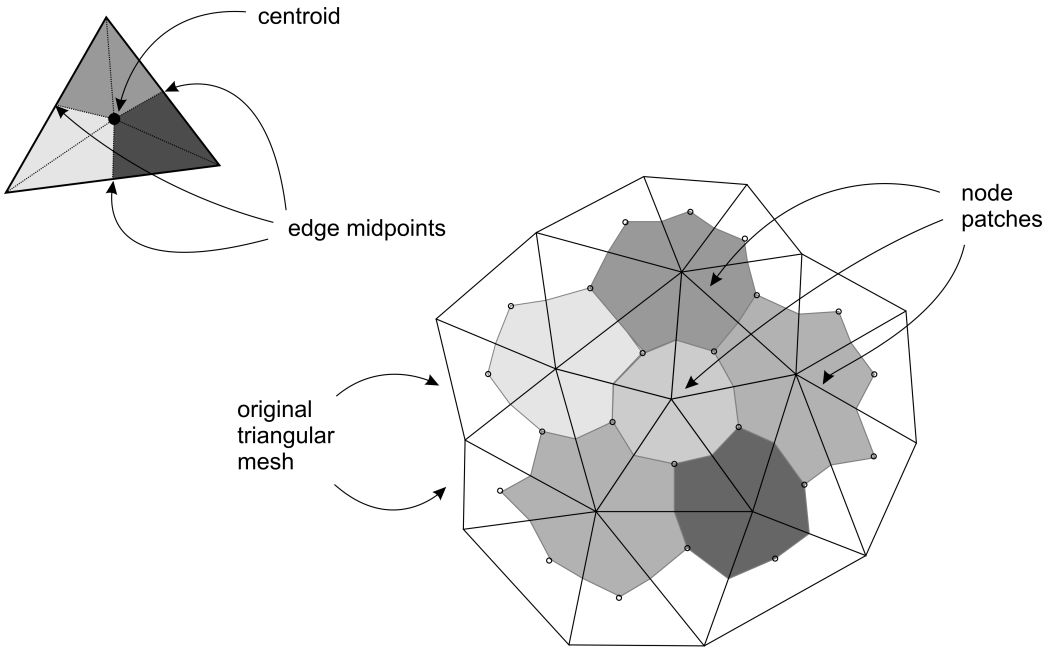


Figure 3.4: Creating node-patches from a mesh of planar triangles: the number of sub-triangles in a node-patch is double the number of triangles around the original vertex.

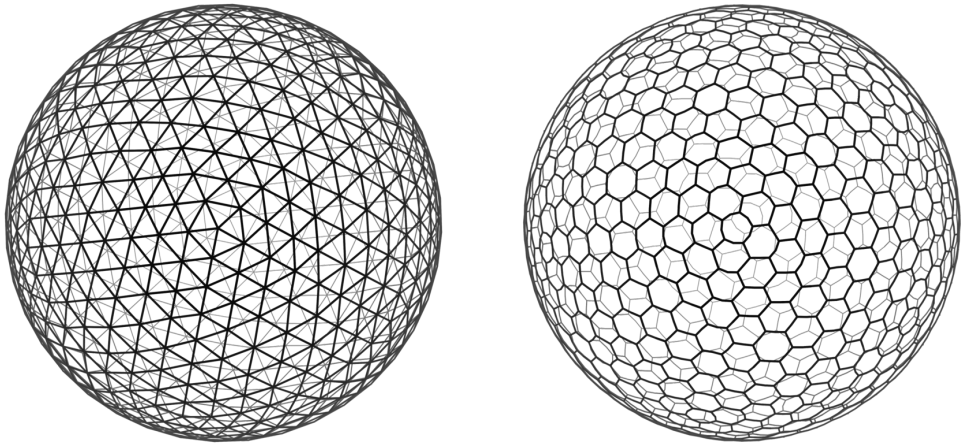


Figure 3.5: 642 vertex sphere (subdivided icosahedron) meshes as 1280 planar triangles and as 642 node-patches.

3.2.2.3 Numerical Integration by Quadrature

Quadrature is the numerical method used to approximate a continuous integral of a given function $g(x)$ by a discrete weighted sum of the function evaluated at a number (Q) of special quadrature points (or "abscissae"):

$$\int g(x).dx \approx \sum_{i=1}^{Q} w_i.g(x_i) \tag{3.1}$$

where w_i is the weight of the i^{th} quadrature point, x_i. Mathematically quadrature rules work by assuming the integrand follows some polynomial function. If the integrand can be well approximated by a polynomial, then the quadrature will give an accurate estimate of the true value of the continuous integral. There are many different families of quadrature rules, most of which are defined for 1-dimensional functions, and the choice of quadrature "family" is closely related to the type of function being integrated. Within each family there are increasing accuracies of quadrature rule: in general the *order* (corresponding to the number of points, Q, in Equation 3.1) of a quadrature rule indicates the number of function evaluations required by that rule, and the *degree* indicates the highest degree polynomial for which the quadrature rule can provide an exact result. Higher degree rules have potentially higher precision, but generally require more point (are higher order) than lower degree rules. A caveat is that if the quadrature rule is not suitable for your function, then increasing the number of points does not necessarily increase the accuracy of the numerical integration, so care is needed when choosing quadrature rules. Quadrature rules are easily found in the literature, within scientific computing software libraries (such as the GNU Scientific Library), or by consulting Abramowitz and Stegun [129]. Within BEEP we use the symmetric Gauss-Legendre rules of Dunavant [133].

For two-dimensional integrations, such as integrations over a triangular surface element, two 1-dimensional quadrature rules could be used successively: in general such rules are defined over a unit parametric triangle, and the integration can be carried out first over one parameter then over the second parameter. In practice this tends to be somewhat excessive, and more efficient combinations of quadrature points can be found which give the desired accuracy (for example those given by Lyness and Jesperson [134]).

The locations of quadrature points on the standard unit triangle for 4-point and 7-point Gauss-Legendre quadrature are shown on Figure 3.6.

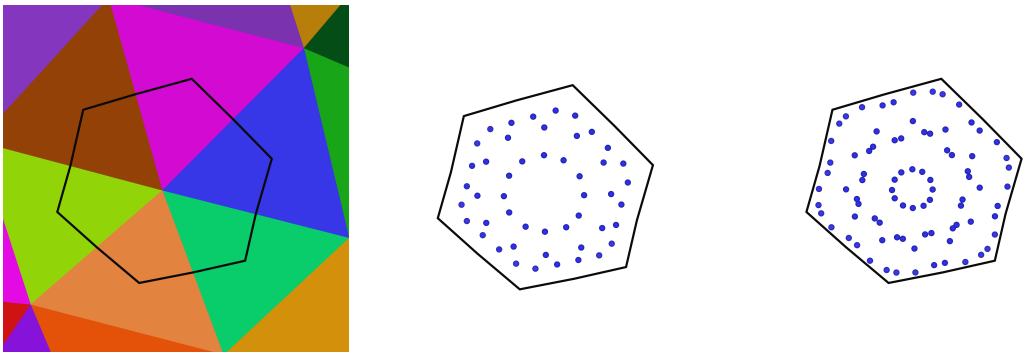


Figure 3.6: For the node-patch depicted (left, black outline of node-patch over original planar triangle mesh), 4-point and 7-point symmetric Gauss-Legendre quadrature rules applied to all sub-triangles comprising the mesh results in the distributions of abscissae shown in blue (middle and right, respectively).

3.2.2.4 Discretization: Galerkin, Qualocation, Collocation

We have now described how we create a discrete surface over which to lay our continuous integral equations (Equations 2.34 and 2.35). Here we introduce three methods for discretizing the equations onto a surface mesh: the Galerkin method; qualocation; centroid collocation. These methods have recently been compared in the context of the BEM for protein electrostatics by Bardhan *et al.* [109].

All three methods convert the continuous problem into a linear algebra matrix-vector equation, with the node-patch values of f and h appearing as elements in a vector of unknowns, to be solved for against a "right-hand-side" vector of known values. The components of the matrix are given by the BEM "kernel functions" A_{pt}, B_{pt}, C_{pt}, D_{pt}.

The discretization methods all aim to convert the BEM integral equations from an expression in terms of an infinitesimal "source point" p into an equivalent expression in terms of a "source node-patch" Γ_p[3]. The next two paragraphs outline the differences between the discretization schemes, which revolves around how the source node-patches (Γ_p, analogous to the source-point p) and "target" node-patches Γ_t are treated in each case.

Source node-patches: low-order vs. high-order sampling The first difference between the methods is in the coercion of the point p into a node-patch: the simplest method is to assume that the node-patch Γ_p can be represented by a single point, such as the centroid of the node-patch or the central vertex, which results in the centroid collocation discretization. Alternatively the node-patch can be represented by a collection of points, such as the centroids of the individual sub-triangles or a set of standard quadrature points

[3]We use the term "source" node-patch (corresponding to integration over some part of the surface Γ_p) to refer to the node-patch *from* which integrations are being carried out; the integrations are carried out *over* the "target" node-patches, (parts of the surface, Γ_t).

over each triangle: this is the case in both Galerkin and qualocation methods. In other words, the three methods are different choices of sampling methodology over node-patches, to represent a node-patch by a single number.

The choice of sampling method has a significant effect on how the discretized system corresponds to the continuous system of equations: low-order sampling (centroid collocation) means that each term in the right-hand-side vector and in the BEM matrix directly represents the value at a single point from the continuous system. Higher-order sampling (used in both qualocation and the Galerkin method) takes the values from multiple points in the continuous system and averages them to form the single value in the discretized system. This requires more computation, but could introduce some useful smoothing of sharp features in the BEM functions[4].

Looking at this in terms of the matrix-vector system of equations, the low-order centroid-collocation method ensures that the solution vector solves the equations with minimum residual error at the collocation points, without reference to any other points on the surface (which may or may not be well satisfied by the solution). The high-order schemes ensure that the solution vector solves the equations such that the *average* residual error over each total node-patch is minimized: if the solution were tested against the continuous equations at some arbitrary point on the surface, it still might not satisfy the continuous set of equations any better than the solution obtained by centroid collocation method *at that point*. Nonetheless the solution obtained by qualocation or Galerkin methods does use more information from the continuous system of equations than the low-order scheme, so intuitively should be "better" overall.

Target node-patches: numerical integration The BEM matrix entries are defined by a surface integration involving the values of f and h over the whole surface. It is simple enough to replace the continuous surface integral with summation of separate integrations taken over each surface node-patch (Γ_t for each node-patch t), with the corresponding values of f and h on each node-patch.

However the Greens functions in the integrals are functions of two positions, r_p and r_t. Once again we are presented with the problem of how to handle the conversion of the source point p (located at r_p) into a node-patch: however we solved that problem in the previous paragraph, and can choose to either carry out the integrations once, from a single

[4]Here we are thinking primarily of beneficial smoothing in the "right-hand-side" functions which might be of use when a node-patch is particularly close to a charged atom, as could happen in an aggressively optimized mesh (i.e. one with very few vertices). Taking a single point on the node-patch could result in a very extreme value being taken of the potential/field, whilst the average over the whole node-patch could be more moderate, leading to a less extreme value emerging in the solution.

representative point on the source node-patch, or take an average of repeated integrations taken from multiple points on the source node-patch.

This leaves the question of how to carry out the integrations themselves over each node-patch Γ_t, which are simply surface integrations over a set of sub-triangles: this can be achieved through conventional numerical quadrature schemes, and the only choice is how many quadrature points (or integration points) should be used to evaluate the integral with acceptable numerical accuracy. This provides the second key difference between the Galerkin, qualocation and centroid collocation methods.

The qualocation method asserts that the surface integrals can be adequately represented by a single quadrature point on each target node-patch (e.g. at the centroid), but that we must use multiple points on each source node-patch. This implies that the values of the BEM integrals are more sensitive to the source-point p than the details of the target node-patches over which the integration is carried out: this seems plausible for the C_{pt} kernel functions which depend on the normal vectors at the source node-patch.

The centroid collocation method already assumes that source node-patches are represented by a single point, their centroid. The order of numerical quadrature for target node-patches can be a single point, which results in a very cheap integration scheme as each integration over a node-patch becomes a single evaluation of the BEM kernel functions, or some higher order scheme, such as 1 point per sub-triangle, or 4 Gauss-Legendre points per sub-triangle. This method implicitly asserts that the values of the BEM matrix are more sensitive to the variation in integrand over target node-patches than to the choice of source-points p, which seems a plausible proposition for the B_{pt} kernel functions which depend on the normal vectors at target node-patches.

The Galerkin method is conceptually both of these methods combined: multiple points on each source node-patch with high-order integration over each target-patch, i.e. multiple quadrature points: this should suit all of the BEM kernel functions, particularly A_{pt} and D_{pt}. Whilst this method offers maximum sampling with maximum accuracy in calculation of each surface integral, the price is a very large number of evaluations of the BEM kernel functions compared to the other two schemes: in practice the Galerkin method is too slow to be practical for biomolecular simulation.

Summary The discretization methods are summarized in Table 3.1 and illustrated in Figure 3.7, which also shows how these concepts relate to the incorporation of the FMM into the BEM.

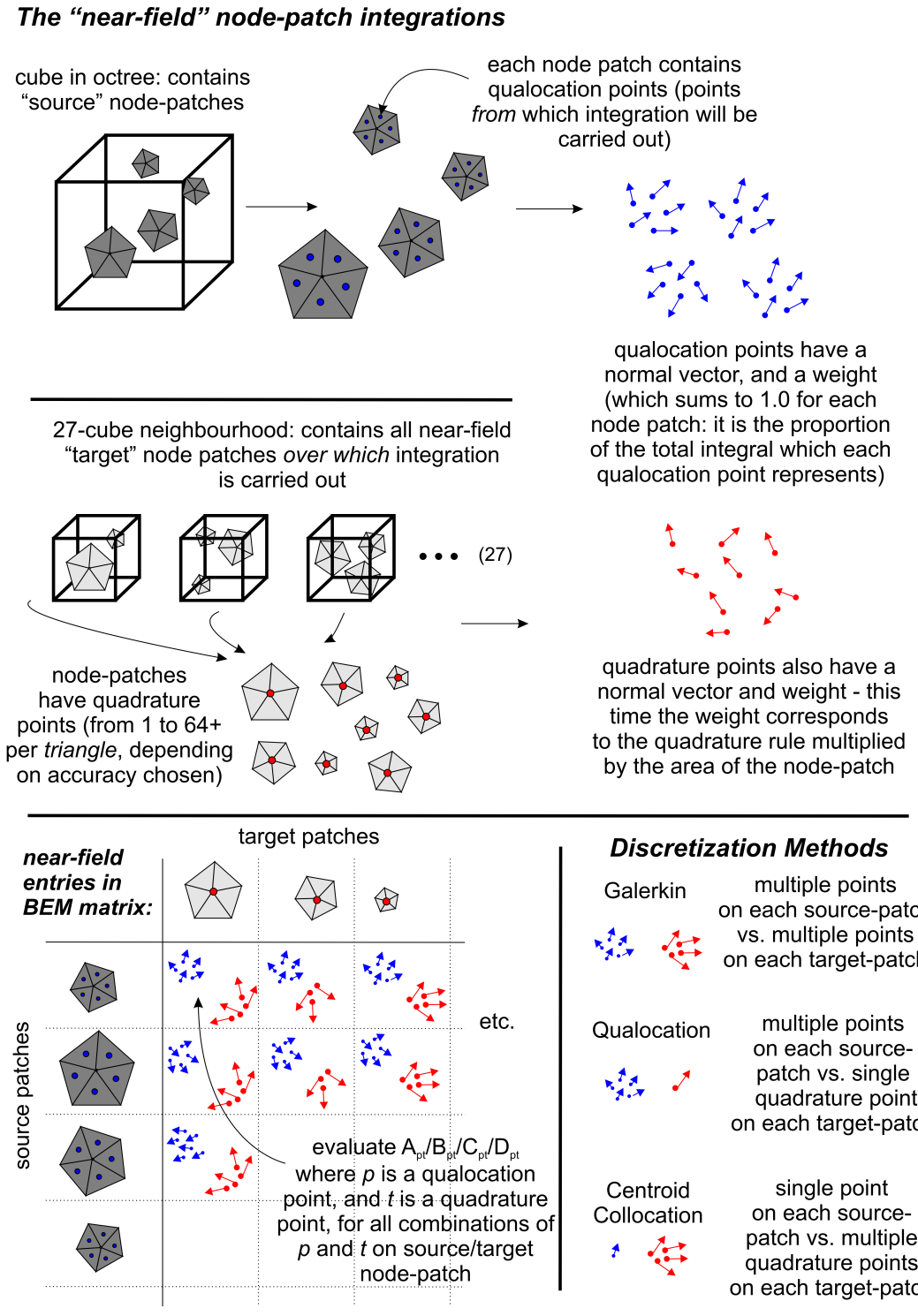


Figure 3.7: illustration of the relationship between FMM cubes, the near-field integrals, and the different possible discretization approaches, which correspond to the choices of how to go about carrying out numerical integration from points on one discretized node-patch element over another (see Section 3.2.2.2).

Discretization Method	Sampling points on "source" node-patches	Order of quadrature rule over "target" node-patches	Comments
Galerkin	Multiple: e.g. 4 Gauss-Legendre points per sub-triangle	Multiple: e.g. 4 Gauss-Legendre points per sub-triangle	Best representation of the continuous system within a discrete framework, but unfeasibly slow if more than a few points used on either source or target node-patches.
Qualocation	Multiple	Single point (centroid of node-patch)	Solution to discrete system minimizes residual error averaged over node-patches. Faster than Galerkin as only single-point quadrature used in BEM integrals, implying that variation in source-point is more important in the BEM kernel functions. Almost as fast as centroid collocation.
Centroid collocation	Single point (centroid of node-patch)	Multiple	Solution to discrete system minimizes residual error only at single points (centroids). Assumes BEM integrals are most affected by variation in position over target node-patch. The fastest option, since only single samples of source node-patches are needed to construct the right-hand-side vector.

Table 3.1: Summary of discretization methodologies.

3.2.2.5 Fidelity

We define the *fidelity* of the solution as the extent to which the solution (values of f and h, found by solving the discretized system) resembles the infinitely-refined continuous solution, which in general is unavailable to use, except for in a few rare analytical cases. The choice of discretization method does not directly affect the fidelity of the final solution: this is controlled by the choice of basis function which defines f and h on each node-patch (we assume a constant value over each node-patch), and the number of node-patches (i.e. the spatial resolution of the mesh).

The discretization method can *indirectly* hamper the fidelity of a given solution by providing discrete values within the matrix-vector system which do not map well to the continuous system of equations. However we would expect that in the limit of a very large number of node-patches the choice of discretization method would become irrelevant, because the node-patches would start to resemble points and a single sampling point for each node-patch

would accurately reflect the value across the whole node-patch (and single-point numerical quadrature would give the same result as higher-order quadrature).

Conversely, deliberately choosing a low resolution mesh with very few node-patches almost certainly results in a solution with limited fidelity. Using very high-order discretization/integration schemes in this case may be a waste of effort: the solution is doomed to be a coarse approximation of the "true" solution which we could obtain with infinite numbers of node-patches, and a very precise solution to a coarse system of equations is not necessarily of any more practical use than a more approximate solution to the same set of coarse equations.

3.2.2.6 The Fast Multipole Method as numerical integration

Within the BEM/FMM we use the FMM to approximate the far-field components in the matrix-vector equation, which avoids the necessity to explicitly calculate and store each element of the BEM matrix. The above discussion of numerical integration over target node-patches is therefore only applicable to the near-field node-patch interactions, whilst the remainder are approximated by the FMM. This leads to a possible discontinuity between the two methods: the FMM represents each node-patch by a single point in the FMM octree, which is analogous to using a single sampling point on each source node-patch and a single quadrature point on each target node-patch. The FMM carries out the numerical integrations using, in effect, the coarsest possible discretization/integration schemes, whilst the near-field may be calculated with much more care and attention. Altering the near-field "horizon" by altering the neighbourhood size in the FMM octree could conceivably introduce artifacts into the solution caused by the sudden change in numerical method at some distance from each node-patch. Fortunately we expect that the dominant terms in the BEM matrix are located in the near-field (since the BEM kernels decrease as $\frac{1}{r}$ at least), so this effect should be minimal in practice for reasonable neighbourhood sizes.

Following this logic, we would expect that the total far-field contribution to each line of the BEM matrix-vector equations to be less important than the near-field terms, which suggests that we may be able to be a little more relaxed about the numerical accuracy of the FMM far-field: recall that the numerical accuracy of the FMM itself is controlled by the number of terms in the multipole expansion, and lowering the number of terms results in much faster performance of the FMM at the expense of numerical accuracy. We will return to this point in Chapter 5.

3.2.2.7 Singular integrals

One of the main problems with implementing the BEM is the presence of near-singular integrals on the diagonals of the BEM matrix: the kernel functions A_{pt}, B_{pt}, C_{pt}, D_{pt} all exhibit singularities when the source point approaches the target evaluation point, as occurs when the source node-patch is the same as the target node-patch. Some practitioners go to significant effort to remove the singularities by analytical means [97, 135], whilst others find that adopting a reasonably high-order quadrature rule with a cutoff region around the singular points suffices to yield a stable numerical result [105].

BEEP uses Gauss-Legendre quadrature with 4 quadrature points per sub-triangle for the integration of the kernel functions over self-patches, which we refer to generally as the "singular integrals". These are carried out as full Galerkin-like integrations, rather than by qualocation or collocation, regardless of the discretization/integration method used elsewhere. This methodology appears to give sufficiently accurate and stable results (which do not change significantly when the order of quadrature is increased on the "singular" node-patches).

3.2.2.8 Node-patches revisited: why do they improve "accuracy"?

We introduced the node-patch above and stated that Lu *et al.* found them to give better "accuracy" in terms of solvation energy for spherical systems, but did not comment on why that may be.

We suggest that one of the reasons for the apparent success of the node-patch shape (compared to planar triangles) is the mesh-regularizing effect of drawing umbrella-shaped patches around each vertex: the impact of slender or unusually small triangles is ameliorated because what would be an "ill-conditioned point" in the BEM matrix, with numerically large coefficients that would be difficult to obtain with high accuracy using our chosen quadrature rules, is replaced by one with more locally-averaged coefficients. This dilutes the impact of a slender triangle within a set of larger, more easily integrable triangles. In other words, the node-patch succeeds because it allows simple quadrature rules to more accurately evaluate the integral functions, and rare but serious errors produced by anomalous triangles are avoided. At the same time the mesh still covers the entire surface (i.e. we haven't simply deleted the error-prone triangles).

Additionally we suggest that the non-planar nature of the node-patches often results in a better constant-element approximation to the true surface solutions. Intuitively it would seem that a node-patch with an amount of effective curvature would be more likely, in physical reality, to exhibit an iso-value of potential or field than a planar triangle, since

the atomic point charges (which are the primary determinant of the surface solutions) have radial potential functions[5].

3.2.3 Output and post-processing: solvation energy and forces

The outputs of BEEP are the surface potential/field solutions, which can be used for visualization purposes. Alternatively energies and/or forces on the meshed objects can be found by running post-processing scripts over the results.

3.2.3.1 Solvation Energy

As described in Section 2.2.2 of Chapter 2, the surface solutions (f and h) of the BEM equations is mathematically equivalent to a surface charge and dielectric layer which, from the point of view of internal points in the protein, represent the total effect of the external electrostatic environment (i.e. other proteins, the induced reaction field of the solvent, and the effect of mobile ions in the solvent). Mathematically, the electrostatic solvation energy is equal to the total energy of those surface charge distributions interacting with the original point charges representing atoms.

The potential at a point in the interior of a closed mesh is given by Equation 2.11. This expression includes the Coulomb potential due to internal charges. Since the electrostatic solvation energy is the change in energy produced by a transition from uniform to discontinuous dielectric environment, the internal Coulomb terms cancel out, so we can write the electrostatic solvation energy as Equation 3.2. Note that since we earlier defined h to be the external surface normal field component, we must multiply by the dielectric ratio ($\varepsilon_{ratio} = \frac{\varepsilon_{ext}}{\varepsilon_{int}}$) to obtain the internal quantity $\frac{\partial \phi_{int}}{\partial n}$.

$$\begin{aligned}
\triangle E_{solv} &= \frac{1}{2}\sum_k q_k \phi_{int}(r_k) && (3.2)\\
&= \frac{1}{2}\sum_k q_k \left[\oint_\Gamma \left(\phi_{int}(r_t)\frac{\partial G_{kt}}{\partial n} - G_{kt}\frac{\partial \phi_{int}(r_t)}{\partial n}\right) d\Gamma_t\right] && (3.3)\\
&= \frac{1}{2}\sum q_k \left[\oint_\Gamma \left(\frac{\partial G_{kt}}{\partial n} f - G_{kt}\varepsilon_{ratio} h\right) d\Gamma_t\right] && (3.4)
\end{aligned}$$

[5] Admittedly this is only true for convex node-patches, and the situation would be relatively worse for a concavity in the surface heading towards a point charge: however protein surfaces are necessarily more convex than concave.

For the discretized surface, BEEP implements this summation via a combination of numerical quadrature and application of a high-accuracy FMM. The total energetic contribution of each surface element (t) per unit value of f or h can be calculated in the absence of the actual values for f and h, i.e. we can precalculate energy coefficients f_{coeff} and h_{coeff}, according to Equations 3.5-3.7:

$$\triangle E_{solv} = \frac{1}{2} \sum_{t=elements} (f_t.f_{coeff}(t) - \varepsilon_{ratio}.h_t.h_{coeff}(t)) \tag{3.5}$$

$$f_{coeff}(element : S_t) = \sum q_k \cdot \left(\oint_{\Gamma_t} \frac{\partial G_{kt}}{\partial n} d\gamma \right) = \oint_{\Gamma_t} \left(\sum q_k \frac{\partial G_{kt}}{\partial n} \right) d\gamma \tag{3.6}$$

$$h_{coeff}(element : S_t) = \sum q_k \cdot \left(\oint_{\Gamma_t} G_{kt} d\gamma \right) = \oint_{\Gamma_t} \left(\sum q_k G_{kt} \right) d\gamma \tag{3.7}$$

The functions $\sum q_k G_{kt}$ and $\sum q_k \frac{\partial G_{kt}}{\partial n}$ on each element Γ_t are simply the potential and normal component of field produced by the atomic point charges at the boundary element: these can be found by use of the FMM. By using quadrature, the integrals of these functions over each node-patch can be calculated by a weighted sum of the FMM results at quadrature points on each node-patch.

The post-processing script can then evaluate the energy rapidly by simply summing the final values of f and h by the pre-calculated coefficients (after scaling the h terms by the correct dielectric ratio) (Equation 3.5).

3.2.3.2 Forces

Gilson *et al.* [136] provide a useful expression for the volumetric force density acting on a molecule under the Poisson-Boltzmann model, by applying the calculus of variations to a free-energy functional for the Poisson-Boltzmann Equation. The result is:

$$force_{(density)} = \rho E - \frac{1}{2} E^2 \nabla \varepsilon - \frac{1}{2} \varepsilon \kappa^2 \phi^2 \nabla \lambda \tag{3.8}$$

where λ is the exclusion function for ions, and all other symbols have their usual meaning. In order to find the total electrostatic force on a molecule, this expression must be integrated over the volume (Ω). We will describe each of these force components in turn.

qE force & the patch-charge method Taking the first term of Equation 3.8, we have an expression for the volumetric force density resulting from the interaction of the total

electric field acting on the fixed charge density ρ. The total Coulomb force between all charges within any surface mesh must sum to zero, so the Coulombic components of the total electric field E acting on the charge density ρ is zero over the volume of a molecule.

The remaining contribution to ρE comprises the interaction between the reaction field and the k source charges q_k. Since the source charges are point charges, the volume integral can be replaced by a summation:

$$force_{qE} = \int_{\Omega} \rho E . d\Omega = \sum_{k} q_k E_{rf}(r_k)$$

Where $E_{rf}(r_k)$ is the electric field at charges due to the reaction field, at r_k. We will henceforth refer to this total force as the qE force.

In calculating solvation energy, we made use of the potential due to the reaction field at each point charge, ϕ_{rf}. E_{rf} is simply the negative gradient of this function, so we can define the qE force in similar terms to the solvation energy, using the surface solutions f and h. Pre-computation of coefficients for the summation of $E(r_k)$ (similar to the energy coefficients) can also be carried out for the qE force components, giving a fast way to calculate the forces as a post-processing step (at the cost of some pre-processing).

We will refer to this method of finding the qE force component the "patch-charge" method.

The dielectric boundary force (dbf) The term $-\frac{1}{2}E^2\nabla\varepsilon$ in Equation 3.8 describes a dielectric boundary force. The physical meaning of this force is not immediately obvious, but we can deduce that such a force must indeed exist by envisaging a Born ion in which the charge has been placed off-centre. The lack of spherical symmetry means that there is a non-zero qE force acting on the charge (as we will see later in this chapter) which acts towards the boundary, in the direction of the eccentricity of the off-centre charge. In order for the total force on the system to be zero, there must be some balancing force: this is the dielectric boundary force.

The dielectric boundary force depends on the gradient of the dielectric permittivity ($\nabla\varepsilon$) throughout the volume, which has non-zero value only over the dielectric boundary of the molecule. Unfortunately the gradient of the dielectric permittivity is infinite over the infinitesimal thickness of the dielectric boundary. Nonetheless we can find the value of this integral to give a boundary force in terms of a normally-directed surface stress (pressure) over the surface, which can be integrated over the dielectric boundary (see Appendix A) to give the total dielectric boundary force.

The dielectric boundary force has the value [91, 137]:

$$force_{dbf} = \int_{\Gamma} -\frac{1}{2} n \left(\varepsilon_{ext} - \varepsilon_{int}\right) \left(E_{ext} \cdot E_{int}\right) d\Gamma$$

where E_{ext} and E_{int} are the external and internal values for electric field on the dielectric boundary. From our BEM boundary conditions we know that the normal components of E_{ext} and E_{int} are related by the ratio of dielectric constants, whilst the planar (tangential) components of E_{ext} and E_{int} are equal.

The ionic boundary force The final term in Equation 3.8 is an ionic "pressure" ($-\frac{1}{2}\varepsilon_{ext}\kappa^2\phi^2\nabla\lambda$). The ionic pressure is a consequence of ions being attracted by the electric field within the meshed object, but being unable to penetrate the surface, producing a pressure force. This force term is controlled by the function $\nabla\lambda$, which is the gradient of the exclusion function for ions, and corresponds to zero everywhere except the dielectric boundary where $\nabla\lambda = n$, that is, the surface normal vector. The net ionic force is quite easy to calculate as a sum over surface triangles (t):

$$force_{ionic} = -\frac{1}{2}\kappa^2\varepsilon_{ext}\sum_{t} f_t^2 . area_t . n_t \tag{3.9}$$

in which f_t is the mean value of the potential at the vertices of the triangle.

An alternative method for calculating forces: the Maxwell Stress Tensor It is well known that the volumetric force density represented by ρE can be converted into the divergence of a stress tensor, such that [138]:

$$force_{\rho E} = \int_{\Omega} \rho E . d\Omega = \int_{\Gamma} (\nabla \cdot T) d\Gamma$$

where T is the Maxwell Stress Tensor (MST), given by Equation 3.10.

$$T = \varepsilon \begin{Bmatrix} \left(E_x E_x - \frac{1}{2}E^2\right) & E_x E_y & E_x E_z \\ E_y E_x & \left(E_y E_y - \frac{1}{2}E^2\right) & E_y E_z \\ E_z E_x & E_z E_y & \left(E_z E_z - \frac{1}{2}E^2\right) \end{Bmatrix} \tag{3.10}$$

$$T_{ij} = \varepsilon \left(E_i E_j - \frac{1}{2}E^2 \delta_{ij}\right) \tag{3.11}$$

The derivation for this relationship is given in Appendix A. The MST provides us with a mathematical system by which the total force on a bounded volume of charge density can

be represented as an integral of surface stresses: we can integrate the MST over any surface we choose to give the net force acting on the volume enclosed by the surface[6]. However it is of some importance to note that the stress calculated for each surface element is "not real", in that the system of stresses described by the MST are not compatible with real materials. The MST is mathematically equivalent to the volumetric force density, but it is the volumetric body force which corresponds to physical reality, not the Maxwellian stress system (see Appendix A for details).

If we choose to apply the MST to a surface *just inside* the dielectric boundary, we can find the total force acting on the enclosed charges. That is, the MST on the interior of a molecule gives the qE force. We previously found we could calculate this by using the patch-charge method; the MST provides an alternative.

If we apply the MST to a surface *just outside* the dielectric boundary, the BEM literature commonly states that we obtain the total force on the molecule[7] (apart from the ionic boundary pressure) [105]. If we take the difference in terms between the MST applied to the inside and outside of the boundary, we find that the difference in MST terms is exactly the same as the expression for dbf (the dielectric boundary force).

To summarise, the resultant force for the MST taken *internally* on the boundary is the qE force on the molecule, which should give the same result as the patch-charge method. The resultant force for the MST taken *externally* on the boundary is the sum of the qE force and the dbf. The total force on the molecule is the sum of qE force, dbf and ionic pressure.

3.2.3.3 Electric Field

In order to calculate the total force on a macromolecule described by the PBE using the expressions given above, we require the electric field vector E. The output of BEEP gives the variation in the scalar potential ($\phi \equiv f$) over the surface of the molecule and the normal component of electric field ($E.n \equiv h$), which is not quite enough information to give us E directly. We have the normal component of E, but the two remaining planar components tangential to the dielectric boundary must be found from values of ϕ. This presents us with

[6] provided that the core assumptions of the MST remain true: i.e. the volume is an isotropic and uniform dielectric medium.

[7] The BEM literature commonly argues [99] that the total force is obtained because the MST represents the total force acting on the volume which it encloses. We suggest this is somewhat misleading, though (fortunately) it is also correct in this case. The MST derivation assumes a uniform dielectric constant. By applying the MST to the exterior side of the dielectric boundary, where $\varepsilon = \varepsilon_0\varepsilon_{ext}$, we are no longer dealing with a homogeneous dielectric constant, because the dielectric constant of the enclosed volume is $\varepsilon_0\varepsilon_{int}$. In fact this does not matter here because the underlying assumption of uniformity relates to the appearance of the dielectric constant in the derivative form of Gauss' Law (see derivation of MST in Appendix A). The change in dielectric in this case (the boundary) does not coincide with any charge density, so the MST remains valid.

the problem of numerically differentiating the surface potential over the surface. Inaccuracy in our calculation of the planar components of E will lead to inaccuracy in the forces based on E.

Lu *et al.* [99] describe a method for interpolating the electric field from the surface solutions which operates as follows:

- any point on the surface where an electric field is required is located within a planar triangle element, and can be mapped into that triangle in local barycentric coordinates, $r(u, v, w)$ where $u + v + w = 1.0$.

- the normal component of the electric field vector is taken as the weighted sum of the vector quantity hn at the vertices of the triangle (hn at each vertex is the vector representation of the normal component of E, as solved by the BEM).

- the planar (tangential) components of the electric field are calculated from changes in values of f at the vertices. Since we have a flat surface with three scalar values, the value of the gradient of f over the triangle is constant[8]. We can find two vectors describing the electric field by the vector quantities: $(v_2 - v_1)(f_2 - f_1)$ and $(v_3 - v_1)(f_3 - f_1)$ where v_i is the Cartesian coordinates of the i^{th} vertex, and f_i is the corresponding value of potential at that vertex.

- These three vectors describe the electric field, but are not necessarily mutually orthogonal. In order to arrive at a value for electric field vector in Cartesian coordinates, Lu *et al.* solve a 3×3 linear set of equations in which the directions of the non-diagonalized component field vectors are matrix elements, the magnitudes of the field vectors are compiled into a right-hand side vector, and then the unknown Cartesian components of E are found by inversion of the matrix (e.g. using singular value decomposition):

[8] In fact the assumption of a constant-element basis function over the surface within the BEM means that the potential has been modelled as taking three constant values on the triangle (one for each node-patch to which the triangle contributed), and is not a smoothly varying linear function over the surface: however if we follow that reasoning to its conclusion, then that would imply that the tangential component of electric field in each third of the triangle is zero (because f is constant), so the field is entirely described by the normal component. This is in fact the condition for a conducting surface. The only place with planar-components of electric field would be the boundaries between node-patches where the planar components would be infinite. In short, we must assume a linear change in f across the triangles or we get very poor results, despite the assumption to the contrary which we make in discretizing the BEM functions.

$$[\Xi]\begin{pmatrix} E_x \\ E_y \\ E_z \end{pmatrix} = \begin{pmatrix} (wh_1 + uh_2 + vh_3) \\ f_2 - f_1 \\ f_3 - f_1 \end{pmatrix}$$

$$\Xi = \begin{bmatrix} (wn_{1x} + un_{2x} + vn_{3x}) & (wn_{1y} + un_{2y} + vn_{3y}) & (wn_{1z} + un_{2z} + vn_{3z}) \\ (v_2 - v_1)_x & (v_2 - v_1)_y & (v_2 - v_1)_z \\ (v_3 - v_1)_x & (v_3 - v_1)_y & (v_3 - v_1)_z \end{bmatrix} \quad (3.12)$$

3.3 Comparison between BEEP and analytic test cases

There are very few systems for which analytic solutions to the linearized PBE exist, and therefore only a limited number of analytic test cases against which to verify BEEP. However even the relatively simple test-cases can give useful insight into the strengths and weaknesses of the boundary element method, and these appear frequently in the literature [72, 91]. Here we use the Born ion, Coulomb's Law, and the off-centre charge within a Born ion to explore the validity of results calculated by BEEP.

3.3.1 The Born Ion

A description of the Born Ion and a formula for its electrostatic solvation energy is given in Section 1.4.4.2 of Chapter 1. The analytic result for the solvation energy was given by Equation 1.25:

$$\Delta G_{Born} = -\frac{q^2}{8\pi a \varepsilon_0}\left(\frac{1}{\varepsilon_{int}} - \frac{1}{\varepsilon_{ext}}\right)$$

where a is the radius of a spherical cavity containing a charge q.

In BEEP the discretized Born ion is represented by a 642 vertex sphere, as illustrated in Figure 3.5, which was created by repeated subdivision of an icosahedron followed by normalization of the vertex coordinates, giving a reasonably good approximation to a spherical surface. We calculated the Born solvation energy using a variety of the discretization/integration methods discussed above, in order to ascertain the accuracy of BEEP results for the various possible options of numerical method.

Table 3.2 shows the results given by BEEP for this simple system, starting with the analytic solution and a high-detail result using the "best" numerical methods (Galerkin discretization, 16 symmetric Gauss-Legendre integration points per sub-triangle of each node-patch) and then using progressively more approximate methods. In most cases all BEM kernel integrations are carried out explicitly, and the FMM is not used at all (the number of vertices is so small that the $O(N^2)$ algorithm can be carried out in a short amount of

time (i.e. seconds/minutes), even when using the compute-intensive 16pt Galerkin method, without resorting to the FMM). "FMM-equivalent" is a centroid-collocation discretization with single-point quadrature (per node-patch), and should give results equivalent to the FMM, if the FMM had a very large number of multipole terms. In FMM cases, only the singular node-patch integrations are carried out by Galerkin-method Gauss-Legendre quadrature, all other node-patch integrals are approximated by multipole expansions with the stated number of terms.

Method	Solvation Energy (kJ/mol)	error (%)	Force magnitude (kJ/mol.Angstrom)
Analytic Solution	-685.86	0.00	0.00E+00
16pt Galerkin (per sub-triangle of node-patch)	-687.87	0.29	9.57E-05
4pt Galerkin (per sub-triangle of node-patch)	-687.93	0.30	9.50E-05
1pt Galerkin (per sub-triangle of node-patch)	-689.43	0.52	9.57E-05
16pt Quadrature (no special singularities)	-710.14	3.54	1.10E-07
16pt Qualocation (no special singularities)	-697.08	1.64	1.15E-07
16pt Quadrature (+16pt Galerkin singularities)	-694.77	1.30	9.62E-05
16pt Qualocation (+16pt Galerkin singularities)	-682.21	-0.53	9.54E-05
FMM-equivalent (+4pt Galerkin singularities)	-689.55	0.54	9.65E-05
FMM (4 terms, +4pt Galerkin singularities)	-670.67	-2.21	4.34E-02
FMM (9 terms, +4pt Galerkin singularities)	**-687.30**	**0.21**	**7.22E-05**
FMM (18 terms, +4pt Galerkin singularities)	-686.24	0.06	3.59E-06

Table 3.2: The solvation energy and total force on a Born ion ($\varepsilon_{int} = 1.0$, $\varepsilon_{ext} = 80.0$, $a = 1\mathring{A}$, $q = +1e$) computed by BEEP using a variety of discretization/integration methods (using a 642 vertex subdivided icosahedron to model a sphere).

BEEP gives accurate results for the Born Ion system for all Galerkin discretization methods. Centroid collocation and qualocation methods perform less well when no special care is taken for the singular self-patch integrals (i.e. when they are treated by the same qualocation/collocation/quadrature rules as other node-patch integrals). The results of centroid collocation and qualocation are both much improved when the singular self-patch integrals are computed using the Galerkin method. Qualocation does indeed appear to give better results than centroid collocation, as found by Bardhan *et al.* [109].

In summary, BEEP gives results with less than 1% error for the solvation energy of the Born ion system for all Galerkin discretization methods. Centroid collocation and qualocation methods perform less well when no special care is taken for the singular self-patch integrals (i.e. when they are treated by the same qualocation/collocation/quadrature rules as every other node-patch integral), and only give "acceptable" accuracy when those singular patch integrals are carried out using the Galerkin method. Qualocation gives more accurate results than centroid collocation, as found previously by Bardhan *et al.* [109].

Somewhat surprisingly coarser approximations using a single-point quadrature rule, and the corresponding FMM tests, give results as accurate or better than higher-order quadrature using either centroid collocation or qualocation. The explanation appears to be that

Salt concentration (mM)	κ (Å^{-1})	Analytic Solvation Energy (kJ/mol)	BEEP Solvation Energy (kJ/mol)	Error (%)	Force magnitude (kJ/mol.Angstrom)
0	0.000	-685.86	-687.93	0.30	9.50E-05
50	0.074	-686.45	-688.53	0.30	9.16E-05
100	0.104	-686.68	-688.75	0.30	9.02E-05
150	0.127	-686.84	-688.91	0.30	8.91E-05
200	0.147	-686.97	-689.04	0.30	8.81E-05
250	0.164	-687.08	-689.15	0.30	8.75E-05
300	0.180	-687.18	-689.25	0.30	8.66E-05
350	0.195	-687.27	-689.34	0.30	8.61E-05
400	0.208	-687.35	-689.42	0.30	8.55E-05
450	0.221	-687.43	-689.50	0.30	8.50E-05
500	0.233	-687.50	-689.57	0.30	8.43E-05

Table 3.3: Solvation energies and residual force results for unit sphere (642 vertices), with varying ionic strength, calculated using the Galerkin method (4pts per sub-triangle).

the qualocation method under-estimates the magnitude of the energy, and the centroid-collocation method over-estimates it: using a balanced approach of either method (i.e. any Galerkin method, including using a single-point per node-patch) gives a virtuous cancellation of these two "errors", resulting in a value very close to the analytic solution.

The FMM cases (with 9 terms or more in the multipole expansions) does not appear to hamper the accuracy of the overall BEM producing a very good approximation to the analytic result with very similar accuracy to the "FMM equivalent" case. Here the 3 digits of accuracy yielded by the 9 term multipole expansions are sufficient to approximate the far field integrals as well as any other method. Using fewer terms in the FMM gives an increased error (2% for 4 terms) but using more terms does not reduce the error.

The residual force is very small in all cases, except for the 4-term FMM case. The lowest forces are for those cases where the self-patch singular integrals are *not* calculated. While these coefficients are required for the highest accuracy of solvation energy, they appear to contribute the most to the error in total force when integrated over the surface. The high-accuracy (18 term multipole-expansion) FMM is somewhat anomalous in that it gives a smaller force than any of the explicit integration methods.

3.3.1.1 Born solvation energy in the presence of salt

The solvation energy for a Born ion when the exterior solvent includes mobile ions (i.e. $\kappa \neq 0$) can also be used to test BEEP. The analytical function is less simple to obtain than that for the canonical Born ion and here we use the formulae derived by Kirkwood [139] (see Section A.3 in Appendix A). For all reasonable values of salt concentration BEEP gives a solvation energy very close to the analytic value. The residual forces are of similar magnitude to those for the non-ionic test case.

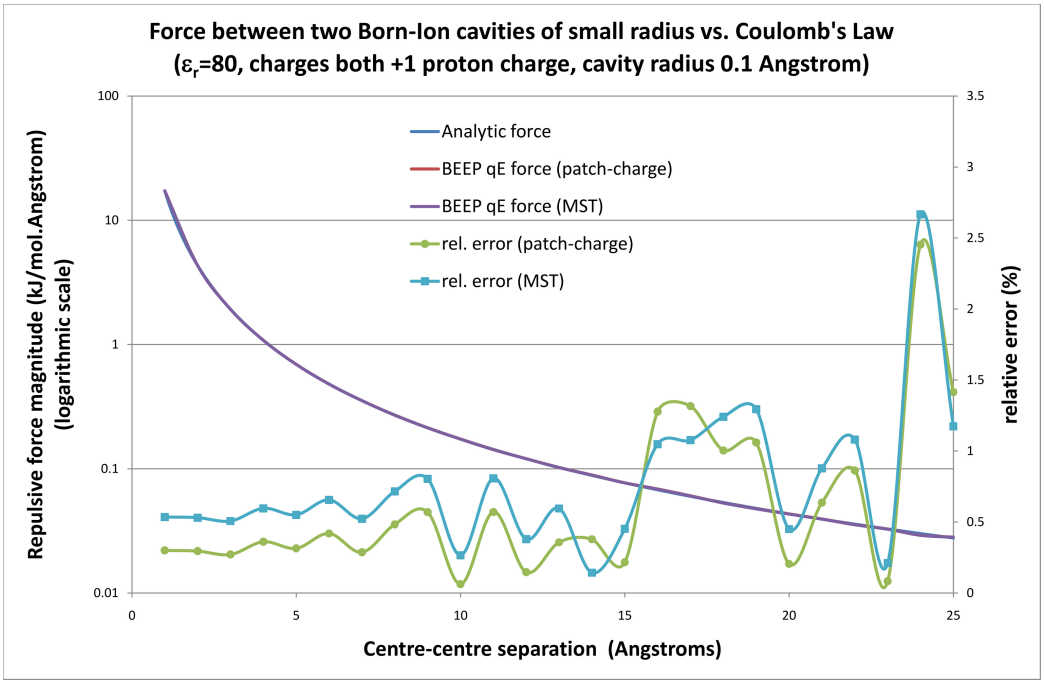


Figure 3.8: Coulomb's Law: analytic vs. BEEP (MST and patch-charge qE methods). The results for forces overlap within the resolution of the plot. The accuracy of the force calculated by BEEP (either MST or patch-charge method) is within a few percent, with relative error generally increasing as the absolute magnitude of the interaction becomes smaller.

3.3.2 Coulomb's Law

We can model two point charges in a uniform dielectric using two Born ions with very small radii (e.g. 0.1Å radius, 642 vertex approximate-sphere) and setting both internal and external dielectric constants to the same value. Figure 3.8 shows the magnitude of the repulsive force between the two point charges separated along the z-axis, calculated analytically and also by BEEP (using both the MST and the patch-charge method to find the qE force). This verifies that BEEP gives results consistent with Coulomb's Law. In general BEEP reproduces the analytic solution to an accuracy within a few percent, with the relative error increasing as the magnitude of the force gets smaller[9].

The BEEP results plotted in Figure 3.8 are for just the z-component of the force for one of the two charges in the system (the x- and y- components being theoretically zero).

Table 3.4 gives more detail of results at two separations, 5Å and 24Å (corresponding to points on Figure 3.8 which have large force and low relative error, and small force and high relative error respectively). The table also shows the effect of using a slightly irregular mesh to describe the sphere rather than a perfectly regular subdivision of an icosahedron.

[9]Note that since the "dielectric boundary" does not represent a change in dielectric constant, the dielectric boundary force is zero, so the qE force is the only term required to calculate the total force.

On-axis forces vs. off-axis forces Firstly we note that the off-axis force components are very close to zero when using the regular mesh (at either separation) and the error in the on-axis force is also small in magnitude (though at large separation the analytic force is very small so the relative error is a few percent, rather than a fraction of a percent). The total net force for the pair of charges should be zero (i.e. equal and opposite forces): for the regular meshes, the total force appears close to zero, and the error is entirely in the on-axis components. The net force error is also the same for both patch-charge and MST methods which suggests that the error is in the solution itself, rather than in the numerical method used to obtain the forces. At larger separations this inherent BEM error (arising from the constant element basis functions, the use of the approximate FMM for the far-field, and the total error in near-field numerical integrals) is less than 2% of the maximum force component on either object [10]. If non-zero net force is considered a major problem from a practical point of view then this relatively small error could be removed by subtracting half of the net force from that calculated for each charge (which would worsen the maximum individual force error, but only by a small amount).

Irregular spherical mesh Most test-cases in the literature are carried out on a subdivided icosahedron, which apart from resulting in a mesh which looks quite spherical, gives a high degree of symmetry in the vertex coordinates. Meshes that describe proteins are much less regular. We created a less-regular spherical mesh to investigate possible masking of errors by the highly regular meshes.

We randomly perturbed the coordinates of each vertex by taking three random numbers (per vertex) from a Gaussian distribution chosen to give small deviations relative to the vertex spacing ($\delta \sim \mathcal{N}(0, \frac{1}{8}\sqrt{\frac{4\pi}{642}})$), after which vertices were renormalized to lie on the spherical surface. The difference between the original regular mesh and the "noisy" mesh is not visible by eye (both meshes appear identical to Figure 3.5).

For the patch-charge method the off-axis force errors increase dramatically from effectively zero to a magnitude comparable to the error in on-axis force (Table 3.4). The net system force also increases with the off-axis forces now similar to the on-axis values. The increased errors are produced by the "un-cancelling" of noise in the numerical method as a whole: errors produced by the surface integrals and their discretization which previously

[10] from Figure 3.8 the individual forces on each charge are not the same for the patch-charge method and the MST: for them to give nearly exactly the same net force means that the error using each method cancels out over both charges, except for the on-axis error: either this remaining small force is the result of cumulative error in the underlying BEM/FMM method (used to arrive at the solutions for f and h), or the two force-calculation methods (patch-charge and MST) are error prone along the z-axis to the exact same degree (not related to underlying inaccuracies in f and h). Given the regularity of the mesh (which as we shall see causes beneficial cancellation of errors) the latter cannot be ruled out. This distinction is largely irrelevant: the important thing is that the net force is very close to zero.

	Forces on Charge A (kJ/mol.Angstrom)			Forces on Charge B (kJ/mol.Angstrom)			Max individual on-axis force magnitude error (% of analytic)	Off-axis relative error (as % of analytic on-axis force magnitude)	Total net system force (kJ/mol.Angstrom)			Net force error (as % of analytic on-axis force magnitude)
	x (off-axis)	y (off-axis)	z (on-axis)	x (off-axis)	y (off-axis)	z (on-axis)			x (off-axis)	y (off-axis)	z (on-axis)	
5 Angstrom separation												
analytic	0	0	-0.69212	0	0	0.69212			0	0	0	
regular mesh: patch-charge	-4.7E-16	2.3E-15	-0.69236	-1.4E-15	2.5E-15	0.69272	0.1	0.0	-1.9E-15	4.9E-15	0.00037	0.1
regular mesh: MST	-1.2E-15	-8.3E-16	-0.69072	-4.7E-16	-1.2E-15	0.69109	0.2	0.0	-1.7E-15	-2.1E-15	0.00037	0.1
noisy mesh: patch-charge	-0.00078	0.00026	-0.69258	-0.00051	0.0003	0.692	0.1	0.1	-0.00128	0.00056	-0.00059	0.2
noisy mesh: MST	-0.01026	-0.00142	-0.70841	-0.01002	-0.00144	0.67266	2.8	1.5	-0.02028	-0.00287	-0.03574	6.0
noisy mesh: corrected MST	-0.00052	5.44E-05	-0.6906	-0.00029	3.87E-05	0.69047	0.2	0.1	-0.00081	9.31E-05	-0.00014	0.1
24 Angstrom separation												
analytic	0	0	-0.03004	0	0	0.03004			0	0	0	
regular mesh: patch-charge	-1.9E-15	3E-15	-0.02941	-9.5E-16	3.2E-15	0.02992	2.1	0.0	-2.8E-15	6.1E-15	0.00051	1.7
regular mesh: MST	-8.3E-16	4.93E-16	-0.02934	2.4E-16	-1E-15	0.02985	2.3	0.0	-5.9E-16	-1E-15	0.00051	1.7
noisy mesh: patch-charge	-0.00078	1.3E-06	-0.03007	-1.4E-06	-4.7E-05	0.02998	0.2	2.6	-0.00078	-4.5E-05	-9.1E-05	2.6
noisy mesh: MST	-0.01028	-0.00171	-0.04757	-0.00949	-0.00176	0.01234	58.9	34.7	-0.01977	-0.00347	-0.03523	135.0
noisy mesh: corrected MST	-0.00055	-0.00023	-0.02976	0.00024	-0.00028	0.03014	0.9	2.0	-0.0003	-0.00051	0.00038	2.4

Table 3.4: Coulomb's Law for two separations: force components, errors, correction of MST results (see main text)

cancelled out serendipitously (or, rather, due to the high symmetry present in the uniformly subdivided mesh) no longer cancel. Interestingly, the maximum individual force error can be seen to reduce when using a slightly noisy mesh, however this is most likely coincidental. Errors due to asymmetry conveniently happen to be similar in magnitude and opposite to the inherent BEM/discretization errors (i.e. those sources of error which produced the small net force even when using the regular symmetric mesh), but there is no obvious reason why this should always be the case, so the apparent improvement is not likely to be generally applicable.

Self-force error in the MST with irregular meshes The qE forces determined by the Maxwell Stress Tensor method become rather inaccurate, and are at least an order of magnitude worse than the forces calculated by the patch-charge method: the forces no longer look close to being equal and opposite, and the net force on the whole system is a significant fraction of the force on each charge. The errors in the on- and off-axis forces are roughly independent of separation between the charges, and consequently at large separation the proportional error in the net force is very large (135% of individual forces).

These "constant" errors suggest that there is a numerical problem in solving the MST on the mesh. Indeed if we calculate the MST force for a single Born ion (noisy mesh, matching internal and external dielectrics of 80), we find that the force calculated using the MST is non-zero. This artificial "self-force" can be used as a constant-offset or "zero-correction" in the MST calculations for the two-component system. The change in force, relative to the isolated case, is included in Table 3.4 as the "corrected MST" and we can see that the accuracy again becomes comparable to that achieved by the patch-charge method.

The apparent failure of the MST method for such a simple system is ominous, as it makes clear that the numerical differentiation to obtain the electric field (E), described in Section 3.2.3.3 leads to a significant amount of cumulative error, at least when using it to calculate qE forces. Fortunately it seems we can characterise the error for the trivial case of an isolated mesh (which should give zero net force) and use that to cancel some of the error from processing of a more complicated configuration. This does, however, require an additional computation of each isolated meshed object.

For the qE force we can avoid this error in the MST by using the patch-charge method instead of the MST. No such alternative exists for the dielectric boundary forces or ionic pressure, which are not present in this uniform dielectric test case. These terms will contribute additional error, especially since the dielectric boundary force is based upon surface integration of E. Consequently this apparently simple example leads us to predict that some

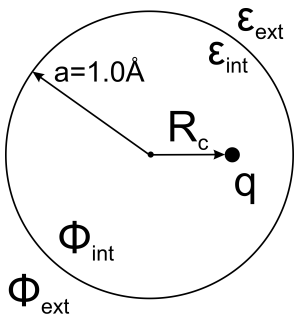


Figure 3.9: Off-centre charge in spherical cavity with unit radius (1Å) (eccentricity denoted R_c).

correction of the dielectric force may be required for realistic models of proteins. This will be discussed further in Chapter 5.

Note that it is not our solution of f or h, or BEM/FMM and numerical integration methods that is at fault in this simple test case using the MST, but the means by which we turn the surface results into useful quantities like intermolecular forces. These problems exist even in the case of simple spherical meshes, however we are unable to see them when using perfectly regular subdivided icosahedra. We recommend that when testing a numerical method for solving the PBE against analytic results, irregular meshes should be used to gain a more realistic view of the limitations of the method.

3.3.3 Off-centre charge in cavity (asymmetric Born Ion) with ionic screening

We have seen that the spherical Born Ion model does not challenge the numerical accuracy of the BEM, and the Coulomb Law is reproduced accurately by the patch-charge method of calculating the qE force. The off-centre charge within a spherical cavity (Figure 3.9) is a much more interesting problem, and introduces us to the effect of charges approaching a dielectric boundary. The formula for calculating the solvation energy of a non-central charge within a spherical cavity with external ion effects is given by Hill [140] and builds on earlier work by Kirkwood [139]: the formulae are reproduced in Appendix A.

The off-centre charge brings us slightly closer to representing the type of charge-boundary interaction which we might expect for real proteins, where boundaries are certainly not perfect spheres and charges approach the surface rather than being located at the geometric centre of the volume.

Figure 3.10 is a plot of the electrostatic solvation energy for an off-centre unit electronic charge in a cavity of radius 1Å (using the same spherical mesh as used for the Born Ion), with an inverse ionic screening length of 3.0. The dielectric constant within the cavity and

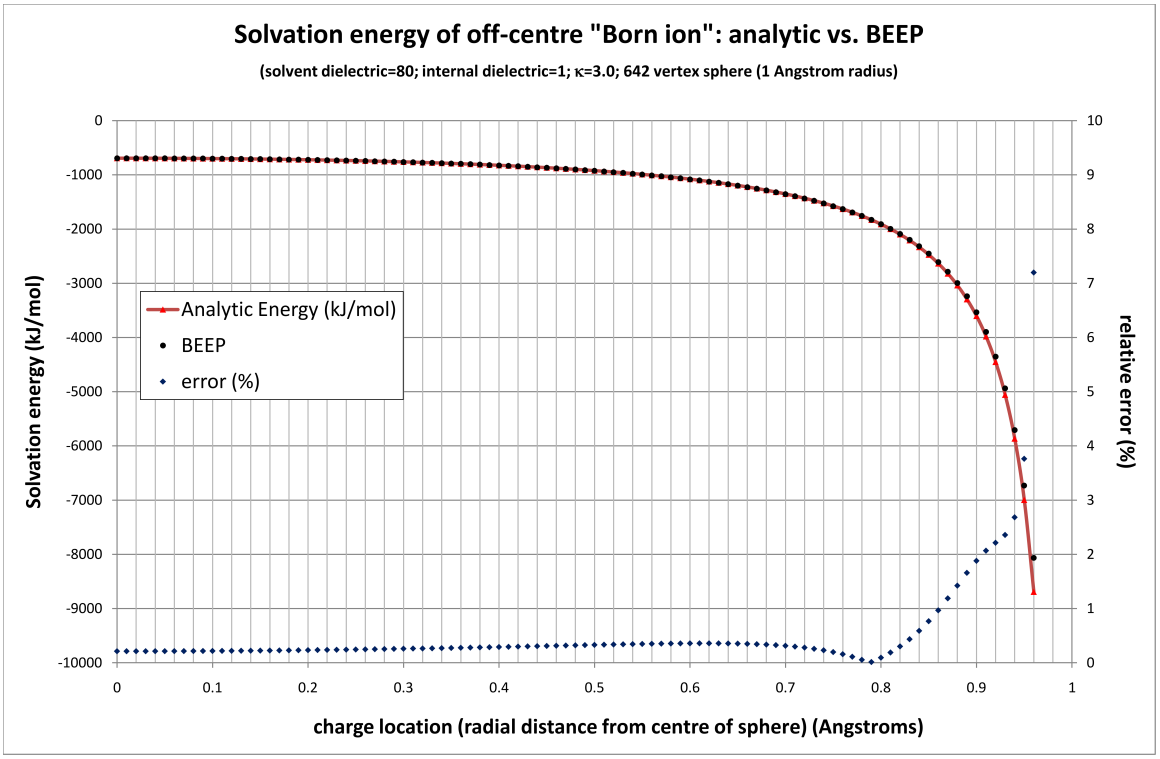


Figure 3.10: Off-centre "Born Ion" results for BEEP

solution were set to 1.0 and 80.0 respectively. There is a decrease in energy as the charge approaches the boundary due to the increased polarization induced by the charge: the charge is attracted to the dielectric boundary, and the attraction becomes stronger as the charge gets closer.

Comparing the analytic curve to that produced by BEEP, we can see that BEEP gives results accurate to within 2% until the charge is within approximately 0.1Å of the boundary, after which the magnitude of the energy steeply drops and the relative error also increases.

It is instructive to look at the forces acting within the system, i.e. the qE reaction field force, the dielectric boundary force (dbf) and the ionic boundary pressure (described in Section 3.2.3.2). The qE force should be exactly counterbalanced by the dbf and ionic forces, giving zero total force for the whole system. Figure 3.11 shows the analytic solution for the qE force component along the centre-charge axis (i.e. radial force component) with the corresponding value calculated by BEEP, using the Maxwell Stress Tensor (MST) (applied on the *inside* of the dielectric boundary) and using the patch-charge summation. The dbf and the ionic force component are also plotted on Figure 3.11. Note that the ionic force is small compared to the other force terms, but of a comparable magnitude to the overall net force (both of these quantities have been plotted on the right-hand axis for greater clarity).

Looking at Figure 3.11, clearly the two estimates for qE force are not equal, and neither

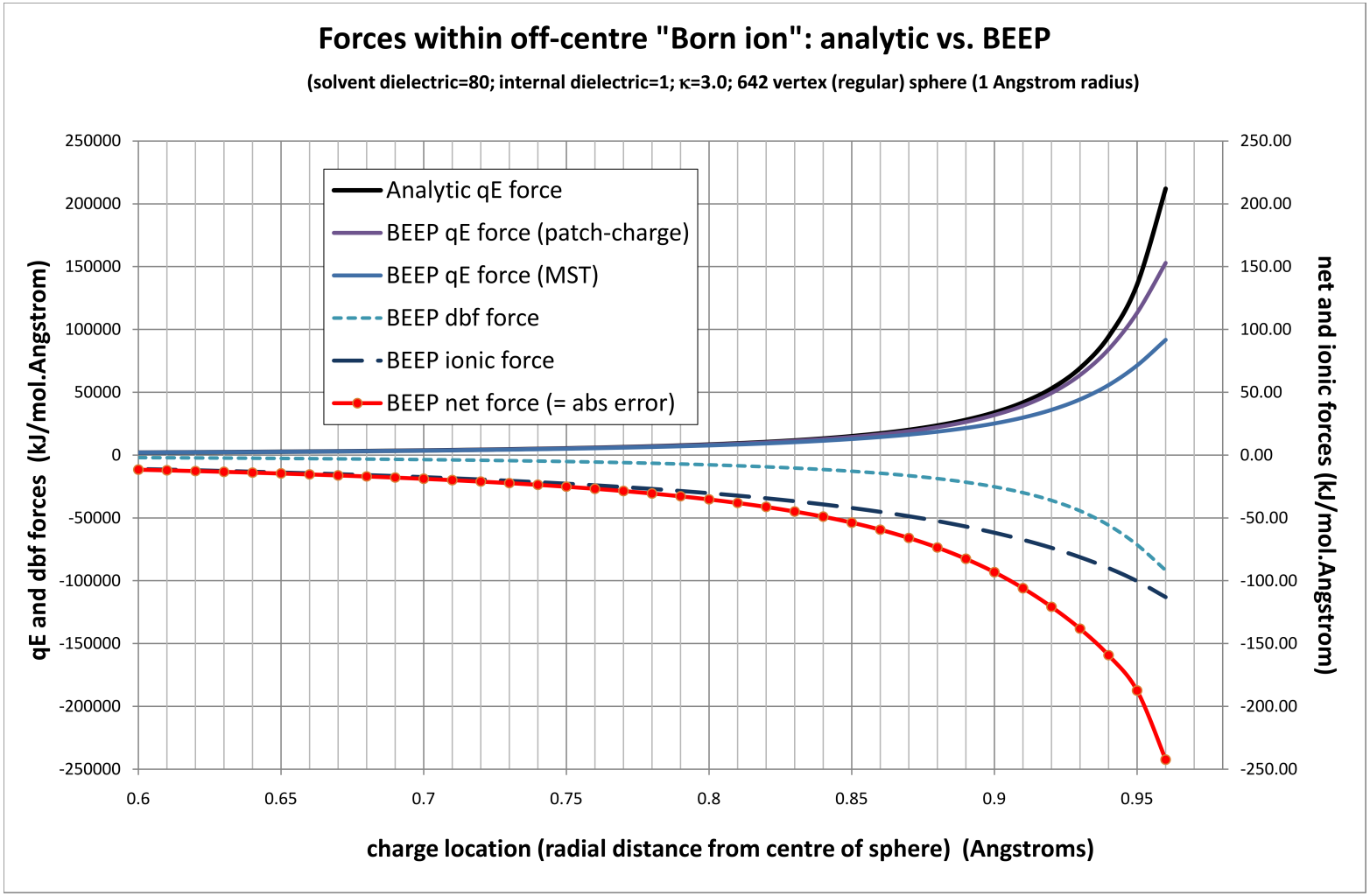


Figure 3.11: Forces on off-centre charge in dielectric cavity: qE and dbf forces are plotted against the left-hand axis, whilst ionic boundary force (navy blue long-dashed line) and the net force (MST+dbf+ionic) are plotted against the larger-scale right hand axis for greater clarity.

coincides perfectly with the analytic value for even quite mild charge offsets. Of the two methods for calculating qE forces, the patch-charge method is closer to the analytic value. However, the value for dielectric boundary force is almost exactly equal and opposite to the value for qE force calculated via the MST. This suggests that the dielectric boundary force contains approximately the same magnitude of error as the qE force calculated by the MST. The net force calculated by combining the MST qE force, the dbf and the ionic force is much closer to zero than that calculated using the patch-charge qE forces[11].

The error in MST qE force and the dielectric boundary force arise from the same source: uncertainty in the calculation of the electric field vector E, which must be obtained from the BEM results by numerical differentiation. Although the Coulomb's Law test above showed that this estimation of E is not very accurate, it appears that the inaccuracy tends to largely cancel out when taking the combination of internal MST and dbf components. We conclude that the net force found in this way is somewhat suspicious: neither the qE force found using the MST or the dbf are anything close to truly accurate for charge eccentricities beyond about 0.5Å, and the relative accuracy of the net force is due to convenient cancellation of errors over the entire surface. In less regular meshes, we might suspect that the errors inherent in the estimates for electric field vector may not cancel so conveniently: indeed in Chapter 5 we will find that for real protein surfaces the cancellation of net force is somewhat less fortuitous than that illustrated here for a sphere.

Scaling the spherical results to put them in biological context These results can be scaled to put them into a more biological context: assuming that the closest and largest charge to any dielectric boundary will be a hydrogen atom of radius 1Å carrying a net charge of +1 proton, let us scale the cavity up to 10Å radius, and the charge location 9Å from the centre. The inverse screening length of $3.0\mathring{A}^{-1}$ then scales to $0.3\mathring{A}^{-1}$ (i.e. a screening length of approx. 3.33Å, which corresponds to about 1M NaCl solution: admittedly somewhat extreme for our hypothetical protein, but this gives us a worst-case accuracy for the ionic forces. If dimensions are increased by a factor of 10, then the energies and forces on the above graphs decrease by a factor of 10 and 100 respectively: therefore the error in energy would be around 2% (an absolute error of approx. 6 kJ/mol), and the error in force would be around 1.0 kJ/mol.Å, which (referring back to the Coulomb Law plot in Figure 3.8) corresponds to the force between elementary point charges separated by approximately 4Å of high dielectric medium ($\varepsilon_r = 80.0$).

[11] As we noted previously, the combination of qE force calculated by the MST on the *inside* of the dielectric boundary, plus the dbf, is exactly the same as calculating the force according to the MST *outside* of the dielectric boundary.

This level of error seems quite high for a single charge in a protein and the total error for many charges would be expected to be larger still. However on the positive side this magnitude of error assumes that the total number of mesh points for the protein corresponds to that used here for the unit sphere: i.e. 642 vertices for the whole surface, whereas in practice a real protein may have a denser mesh than this in order to capture the detail of the molecular surface. This issue is discussed further in Chapter 5.

3.4 Conclusions

We have implemented a working BEM/FMM linear PBE solver called BEEP. In the process of creating BEEP we ported an FMM implementation into C++ to allow simpler parallelisation (which we will discuss in the next chapter), and which is of some use to the scientific community as a stand-alone piece of work (our port includes a number of minor enhancements over the original Fortran code, such as additional accuracy options which were absent in the original code). BEEP extends the capabilities of existing BEM solvers by allowing variable dielectric constant between proteins (this was previously suggested by Lu *et al.* [103] however to our knowledge has never actually been implemented in code, presumably due to the extra 50% cost in computational effort this entails).

BEEP solves the analytic test cases correctly and for the simple spherical models here produces quite accurate values for energies, though it is clear that the solutions become less precise as charges approach the dielectric boundary (for a given mesh density). We have shown that the extraction of force components from the BEM surface solutions can be carried out using either a patch-charge or by use of the Maxwell Stress Tensor, but that the latter is less accurate unless some simple error-cancelling is applied. In the case of an off-centre charge in a low dielectric cavity, it seems that the error in the dielectric boundary force conveniently cancels a large degree of the error in the qE force as calculated using the MST. We conclude that this is due to both of these force components being calculated from the electric field vector, which is apparently not obtained with sufficient accuracy (or perhaps sufficient resolution) by the numerical differentiation method described in this chapter to give a perfect zero-sum force over a spherical surface. The implications of this conclusion, in the context of working with irregular protein surfaces rather than the simple spheres tested here, will be discussed further in Chapter 5.

As a final comment to conclude this chapter, we note that BEEP is rather slow at solving systems even with moderate numbers of vertices, and as presented is not suitable for large-scale simulations of multi-protein systems where the electrostatic energies and/or intermolecular forces are needed at each time step. It was obvious very early in the conception

of BEEP that some parallelisation would be vital: our efforts to improve the performance of BEEP are the subject of the next Chapter.

Chapter 4

Parallel BEEP: multi-CPU and GPU acceleration

4.1 Outline of this chapter

Parallelisation is the process of converting a program which runs on a single processor to carry out a specific task into a program which runs on multiple processors, with the aim of carrying out the same task in reduced time.

This chapter describes the methods we have used to improve the performance of BEEP through two approaches to parallelisation. Firstly we have parallelised BEEP to run on multiple compute-nodes (as found in the now widespread high performance compute clusters), which results in the program "BEEPp" (i.e. "BEEP parallel"). We have also parallelised BEEP to exploit the recent advances in general purpose graphics processing unit (GPGPU) programming, in which the floating-point calculation capabilities of the GPUs is harnessed as a parallel co-processor. The GPU accelerations apply to both BEEP and BEEPp, so the program can give maximum performance for whatever computer hardware is available: multiple hosts or a single desktop, with or without GPU acceleration.

We begin by investigating the performance of the non-parallel BEEP which will give an indication of where the program can be improved. Then we give a short overview of parallel programming methodologies in general, before describing the parallelisation of BEEPp in a little more detail.

4.1.1 Terminology

Hardware In order to avoid confusion, throughout this chapter we refer to each addressable processor core of a computer as a "processor". Several processors (cores) may reside on the silicon inside a single CPU chip in a modern computer (which may have multiple CPU chips in separate sockets on the motherboard). Each physical machine (motherboard, CPUs, RAM, associated peripheral hardware and network interface) we will refer to as a "compute-node" and these are assumed to be the basic building block of a networked cluster of computers. Unless otherwise stated, processors within a compute-node are assumed to be operating in "symmetric multiprocessing" (SMP) mode in which all processors share the memory space on that compute-node (i.e. processes or threads running on processors can all access the same memory, which is located on the compute-node).

Processes & Threads Processes are used to refer generally to programs running on a compute-node; threads are, in general, a sub-component of a process which are spawned to carry out a specific task (or set of tasks) concurrently (possibly exploiting multiple processors). However, a process always has at least one thread, which is the main thread. It is the task of the operating system to map the large number of processes and their constituent threads which are running on a compute node at any time onto the physical processors available (this includes the operating system itself), and we do not concern ourselves with the details of that here. We will use the term "thread" to refer to a thread of execution (i.e. lines of code being executed) running on a processor, in the context of a larger parallel program. Single-threaded code refers to a process with a single thread, whilst multi-threaded means more than one thread with some consequential requirement on the part of the programmer to manage the potential concurrent execution of those threads.

FMM Flavours There are two distinct types of FMM at work in this chapter:

- the "vanilla" FMM, which is the general-purpose type of FMM as described in Section 2.3 of Chapter 2, for solving potentials due to collections of point charges.
- the "12-fold hybrid BEM/FMM" which is in essence the same thing but adapted such that it carries out the FMM on 12 separate sets of "equivalent charges" simultaneously, as this is what is required to incorporate the FMM into the BEM (as described in Section 2.4 of Chapter 2).

BEEP and BEEPp make use of both of these FMM variants: "vanilla" for calculating the right-hand-side vector of the BEM matrix-vector equations, and the "12-fold hybrid BEM/FMM" within the GMRES iterative loop to solve that matrix-vector equation. In

each case the process of "solving" the FMM involves an upward and downward pass, followed by "evaluation" of the local expansions, plus a summation over the near-field, to yield results. (The details of what happens in the upward and downward pass is given in Sections 2.3.4.1 and 2.3.4.2 of Chapter 2.)

In the context of the FMM (both flavours), the space is hierarchically organised into cubes using an octree.

4.2 Experimental Methods

Here we describe the methods and hardware used for the test cases throughout this chapter. Since these will appear to be somewhat out of context without knowledge of the later parts of the chapter, the reader may prefer to skip this section and return to it for reference later.

4.2.1 Hardware

We used a variety of computer hardware to evaluate the parallel scaling of BEEP and BEEPp.

In our office

- A current-generation GPU workstation: Intel Core i7 950 @ 3.0 GHz (quad-core), with 12GB RAM and dual NVIDIA GTX580 graphics cards.

At the Distributed Computing Group, STFC Daresbury Laboratory

- Cluster of 32 "standard" (or "non-GPU") compute nodes: Intel Xeon E5472 @ 3.0 GHz dual-socket, quad-core (i.e. 8 CPU cores per compute-node), 16GB RAM per node, connected by Infiniband network (Mellanox ConnectX DDR IB HBA MHGH29-XTC Infiniband adapters).

- Cluster of 8 "GPU-accelerated" compute nodes: Intel Xeon E5540 @ 2.53 GHz dual-socket, quad-core (i.e. 8 CPU cores per compute-node), 24 GB RAM per node, connected by Voltaire HCA410-4EX Infiniband network adapters, with a NVIDIA Tesla S1070 server containing 4 GPUs connected to each compute-node.

- AMD-based GPU workstation: AMD Opteron 2376 @ 2.3 GHz, dual socket, quad-core (i.e. 8 CPU cores), with 16GB RAM. The compute-node has three AMD FireStream 9270 cards two of which are inside an external server attached to the compute-node.

4.2.2 Measurements

The systems tested vary according to what feature of the code we are trying to demonstrate. In all cases for measurements of BEEP on a single compute-node, timings were taken by our manually modifying code and adding instrumentation to count the number of clock cycles which elapse around certain large functions of the code. Considering the long execution times of the pieces of code being measured, this method was deemed sufficiently accurate to capture the level of detail we are interested in.

For timing/profiling of parallel code (BEEPp), the "projections" module within Charm++ was used, which internally records very detailed information about which parallel objects are executing entry methods at any given time. The Charm++ projections program then offers visualisation tools to examine the collected data.

4.2.3 Test systems of charges/spheres/proteins

Unless otherwise stated, in all cases the dielectric constant for meshes representing proteins was set to 2.0, the solvent dielectric was set to 80.0 and the screening parameter κ was set to 0.127, corresponding to 150mM monovalent salt concentration. The number of qualocation points was set to 1 per sub-triangle of each node-patch, and the singular self-patch integrals were carried out by the Galerkin method using symmetric Gauss-Legendre (with 4 quadrature points per sub-triangle) integration. 9 terms were used in the multipole expansions for the BEM/FMM.

Linear scaling and profiling For measurements of BEEP performance vs. numbers of meshed objects in Section 4.3, we devised a system of meshed objects intended to mimic real proteins: i.e. realistic numbers of "atomic charges" contained within surfaces meshed to a "reasonable" level detail: we chose spherically meshed objects 15Å in radius with 2562 vertices (approx. 0.9 vertices/$\mathring{A}^2$) containing a random distribution of 250 charges, of magnitude randomly chosen from a uniform distribution between -1 and +1.

The spherically meshed "proteins" where added to the layout within BEEP one by one, the system was "solved" from scratch at each iteration, and timing statistics recorded. Since the results of the BEM calculations themselves were of no interest we terminated BEEP after 2 GMRES iterations. Measurements were taken of execution time for the whole program, and the timing relating to the GMRES iterations were multiplied by 20. The FMM neighbourhood size for the BEM/FMM octree was set to 500.

Parallel "vanilla" FMM The test system for FMM tests in Section 4.6 was a set of randomly placed point charges lying on the surface of a unit sphere, for which the potential and its first derivative at each charge location due to all other charges was calculated using the 18-term FMM (6 digit accuracy), with a screening parameter of $\kappa = 1.0$. The number of charges was 100,000 or 1 million depending on the experiment. The larger problem gives a better view of the scaling properties as the problem induces a greater number of subdivisions in the FMM octree, and the time taken for FMM functions becomes the dominant feature of the timing profiles (rather than problem set-up and compute-node initialisation).

Large BEM problem: 275,000 node patches In Section 4.7 and Section 4.8.3 of this chapter, we use a large system of approximately 275,000 node-patches to examine the scaling of the parallelised BEEP when applied to a substantial problem. The node patches are a $4 \times 4 \times 4$ grid of acetylcholinesterase molecules (taken from PDB structure 1MAH), each meshed with 4297 node-patches, and a separation between molecules roughly equal to the double the maximum radius of the protein (2×40Å). The resulting grid of meshes gives 275,008 node-patches in total. The purpose of the test is to provide a large number of node-patches in a geometry relevant for biomolecular purposes, rather than to determine any particular property of the proteins themselves. (In Chapter 5 we show that 4297 node-patches is actually insufficient to adequately represent such a large and highly charged protein). There is no particular reason why we chose to use a real protein in this case rather than the "mimic" sphere used for the measurements of BEEP linearity. As before, to avoid excessive computation, the results for 20 iterations were extrapolated from the results of 2 GMRES iterations.

BEEP and the FMM neighbourhood size: 51,228 node-patches In Section 4.7.1, Section 4.8.2 and Section 4.8.3 we use a smaller BEM test system of 51,228 node-patches, composed of 2 copies of acetylcholinesterase meshed at higher detail than the previous case, with 25,614 node-patches for each mesh. The proteins were placed in space with their centres separated by 100Å along the z-axis. The results for 20 GMRES iterations are extrapolated from the results of two GMRES iterations. Furthermore in the figures relating neighbourhood size to BEEP execution time, the time taken for explicit integrations and FMM evaluations at higher neighbourhood sizes (the linear regimes of each graph) is extrapolated from data collected at lower neighbourhood sizes and was verified to be correct to within 3% at the neighbourhood size of 3000.

4.3 BEEP performance & the need for parallelisation

Timings throughout this section were measured on the Intel Core i7 workstation in our office (running on a single CPU core) described in Section 4.2.1.

4.3.1 Linear Scaling

The execution time for running BEEP on a system of interacting "Born ion" objects scales linearly with the number of surface vertices (or, equivalently, node-patches), as illustrated in Figure 4.1. The advantage of the BEM/FMM linear algorithm compared to a "naïve" $\mathcal{O}(N^2)$ implementation[1] is clear even for very small systems with a break-even point of around 3,500 node-patches. The exact cross-over point depends on the number of iterations to convergence, the accuracy of numerical integration chosen and (to a lesser extent) the FMM neighbourhood size and the shape of the surface. Nonetheless it is clear that for systems of more than a few thousand vertices the linear method is far superior.

However the time taken to solve a system of several thousand node-patches (which, from the simple results for spherical systems discussed in the previous chapter, is an optimistic estimate for the number of node-patches a real protein will require) is still quite large: on the order of minutes, not seconds, which makes BEEP unsuitable for large scale molecular simulation. In order to allow comparison over system size we have chosen to assume that in each case 20 GMRES iterations results in convergence, whereas in reality we find that the number of iterations appears to depend on some geometric property of the system (e.g. spherical objects reach convergence in fewer iterations than highly irregular objects). However from our experience of the meshes for a test-set of proteins (see Table 5.2 in Chapter 5), we chose 20 iterations as a relatively pessimistic estimate for comparison purposes.[2]

4.3.2 Profiling

Figure 4.2 (which is the same as Figure 4.1, except that the area under the curve is shaded to illustrate which tasks contribute to the execution time) shows that the execution of BEEP is dominated by three tasks: calculation of the right-hand-side BEM vector, pre-calculation of the near-field numerical integrals, and the 20 iterations of the GMRES algorithm. Other program tasks such as reading data files and building data structures to represent the meshed

[1] The naïve implementation would involve direct integration of all BEM matrix terms in Equation 2.36.

[2] Furthermore we have assumed that (apart from singular integrals from one patch over itself) it is necessary to carry out numerical integrations using one qualocation point per sub-triangle of a source node-patch and a single quadrature point per target node-patch. If we resort to single-point qualocation/quadrature (i.e. 1pt Galerkin per node-patch) then execution time decreases; correspondingly if we increase the number of integration points the apparent performance increases.

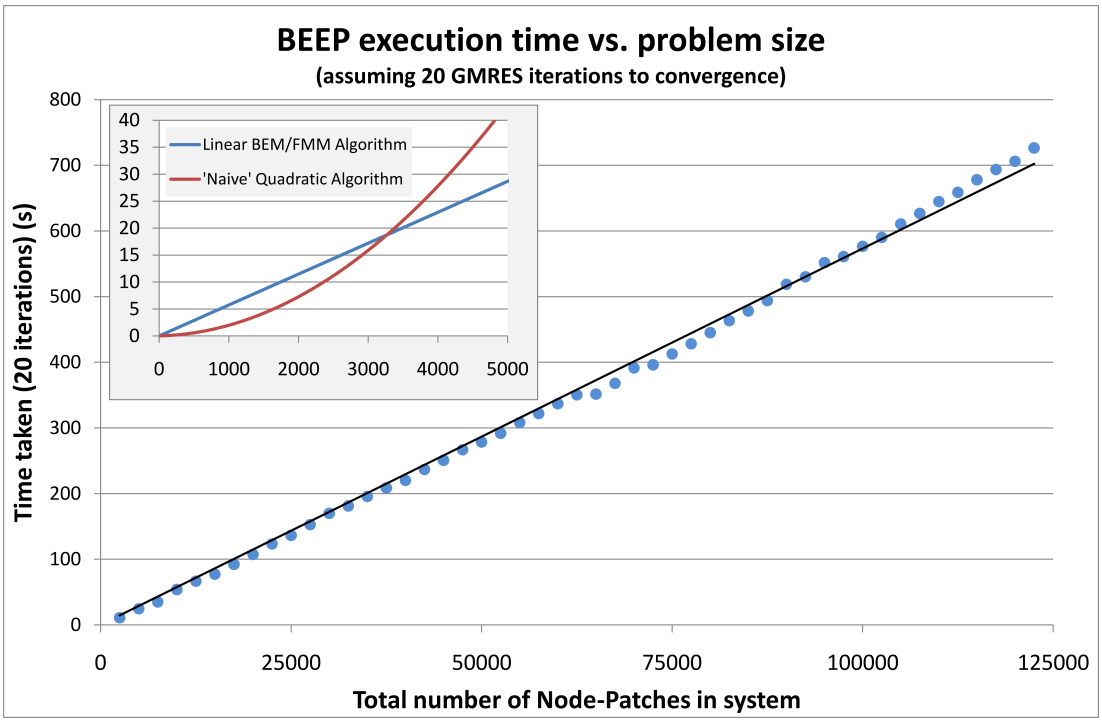


Figure 4.1: The run time of BEEP is approximately linear with respect to the total number of vertices in the system. The inset graph shows the performance of the naive BEM vs. the linear BEM-FMM method (extrapolated backwards from the larger graph) for small numbers of node patches.

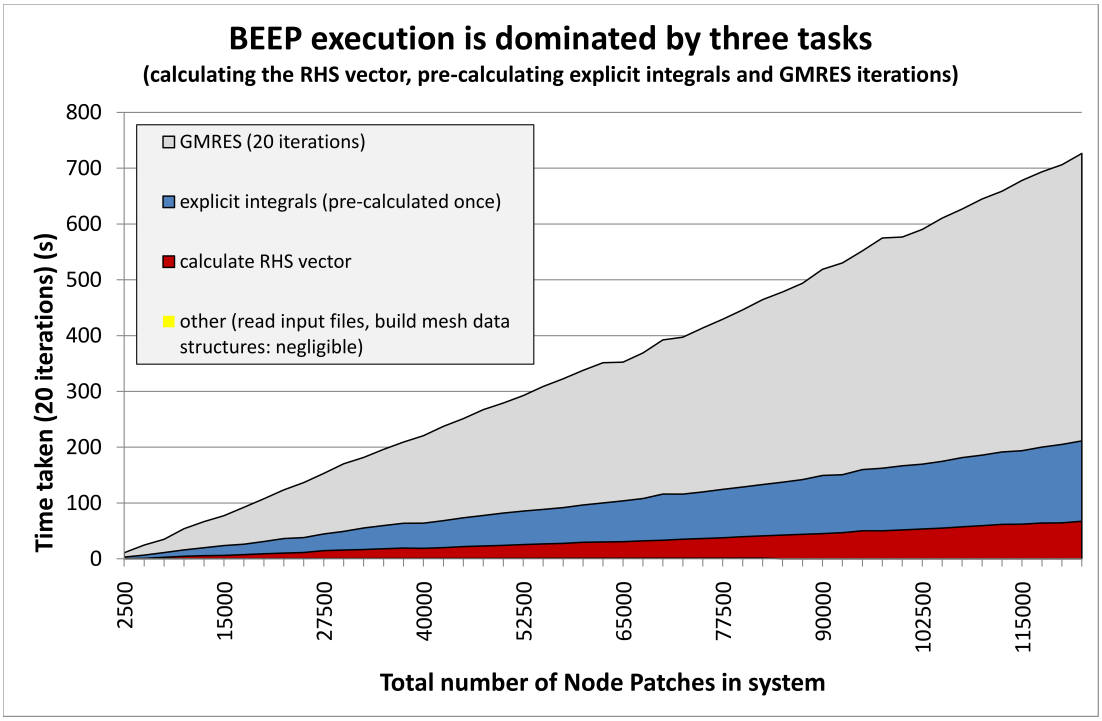


Figure 4.2: Tasks contributing to the execution time of BEEP as a function of system size. The coloured areas under the curve indicate the proportions of execution time taken by each task. The time taken for reading input files and building mesh data structures (yellow) is so small as to be invisible on the graph.

objects take negligible amounts of time in comparison (their contribution is so small as to be invisible on Figure 4.2).

Terms from the numerical integration of the near-field can be precalculated, or computed at each iteration (which would be wasteful in terms of computation, but would save on storage requirements of those terms). In general we choose to pre-calculate the near-field integrals. This process occupies a significant portion of Figure 4.2, with the benefit being that the cost of their usage in each GMRES iteration is approximately zero. The near-field integrations are expected to be readily parallelisable, since they are independent and compute-intensive, requiring little data manipulation or communication.

The calculation of the right-hand-side vector involves use of the "vanilla" FMM (i.e. the canonical FMM used for calculating the potential due to a set of point charges), which from the algorithm description given in Chapter 2 is a little more challenging to break into parallel components. The greatest proportion of time is spent solving the BEM matrix-vector equation through repeated iterations of the GMRES algorithm. Within the GMRES algorithm, we would expect that most computational time is occupied by the 12-fold BEM/FMM procedure, which compute the matrix-vector products required by GMRES. Figure 4.3 shows that this is indeed the case, with the GMRES algorithm itself taking negligible time (a few

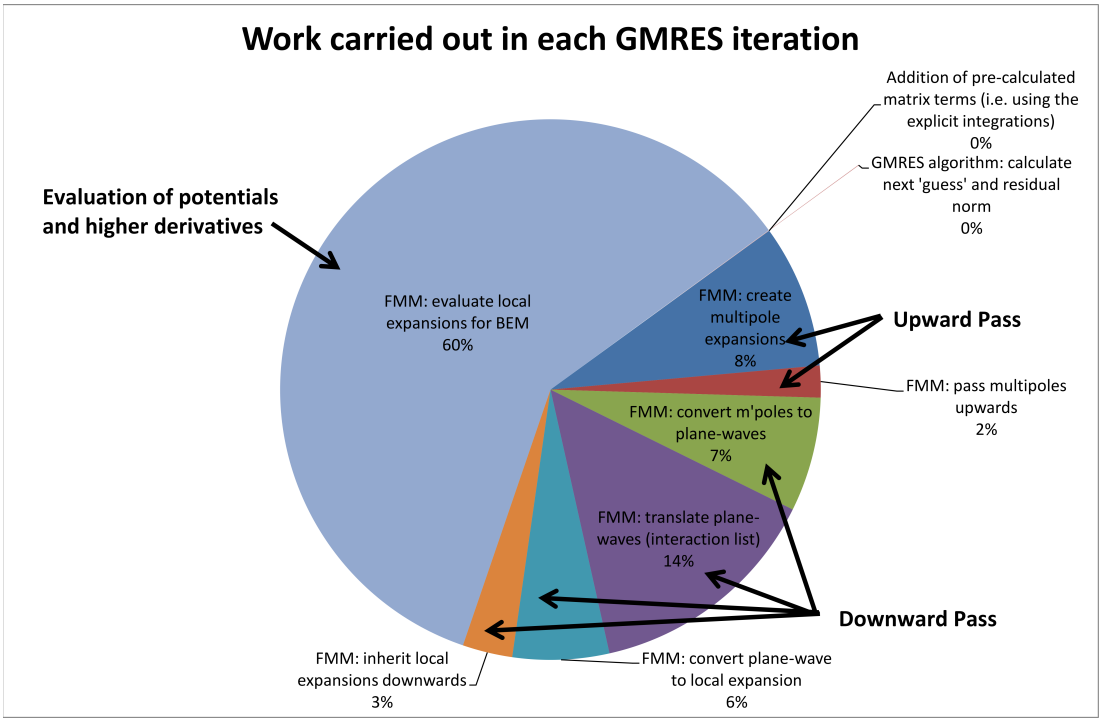


Figure 4.3: Each GMRES iteration can be broken down into a set of constituent FMM operations: the GMRES linear algebra functions (generating new "guesses" for the unknown solution vector and calculating residuals) are negligible in comparison to the much larger FMM tasks. The upward pass takes approximately 10% of the time, the downward pass approximately 30%, and evaluations take the remaining 60% of the time per iteration.

milliseconds).

The 12-fold BEM/FMM algorithm is a more complicated form of the "vanilla" FMM, however the basic processes are the same: an upward pass in which multipoles are formed and passed up the tree; a downward pass involving spatial translation of the multipoles to local expansions (by way of plane-waves) and downward inheritance through the tree; evaluation of the potential and its derivatives at points throughout the tree from the local expansions.

From Figure 4.3 we can see that the majority of the upward pass (which totals about 10% of the work) is in creating the multipole expansions (8%); the most dominant part of the downward pass (total 30%) is in carrying out spatial translations of the plane waves according to the interaction list of each node of the tree (14%), though the combined work of converting to and from plane wave representation (including carrying out the necessary rotations) accounts for almost the same proportion of the work (13%). The remainder of the time spent on each GMRES iteration is the evaluation of local expansions into potentials and higher derivatives (60%).

4.3.3 Amdahl's Law

The well-known relationship "Amdahl's law" states that if we can speedup some *fractional part* of the program P by a factor S then the overall speedup of the program (that is, the ratio of execution time before and after the improvement) will be [141]:

$$\frac{runtime_{old}}{runtime_{new}} = \frac{1}{(1-P) + \frac{P}{S}} \tag{4.1}$$

In the limit that S is a very large number, we are limited to the speedup $\frac{1}{1-P}$. In our case several separate pieces of the algorithm each occupy large chunks of the work, and P for any one piece is not close to 1. Even with a large S the net speedup will be limited following parallelisation of any single task. Thus it would appear that any effort to parallelise BEEP will (unfortunately) involve the entire BEM/FMM process.

4.4 A short introduction to parallel computing

4.4.1 The aim of parallelisation

The aim in parallelising a program is to find a way to break down the computational tasks so as to reduce the execution time linearly according to the number of processors being used. In practice a perfectly linear speedup with respect to number of processors is not achieved for various practical reasons: the program is not entirely composed of parts which can be evenly spread over many processors (i.e. atomicity of program components, for example reading in the configuration files is done by a single thread, and cannot easily be split amongst multiple processors); those tasks which can be subdivided may not happen to balance the computational workload evenly across the processors (load balancing overhead); also there is communication overhead between compute-nodes handling different parts of the program; synchronization overheads where parts of the program need to accumulate some critical result before moving forwards; parallel environment initialisation overheads at the start of the program, and so on. This can be thought of as another manifestation of Amdahl's law: the parts of the program which can be parallelised easily gain a linear speedup, but other parts of the program may take more time due to the increased communication and synchronization overheads: the proportional contribution of non-parallel tasks to the total runtime will increase as processors are added, leading to poor scaling of the overall computation.

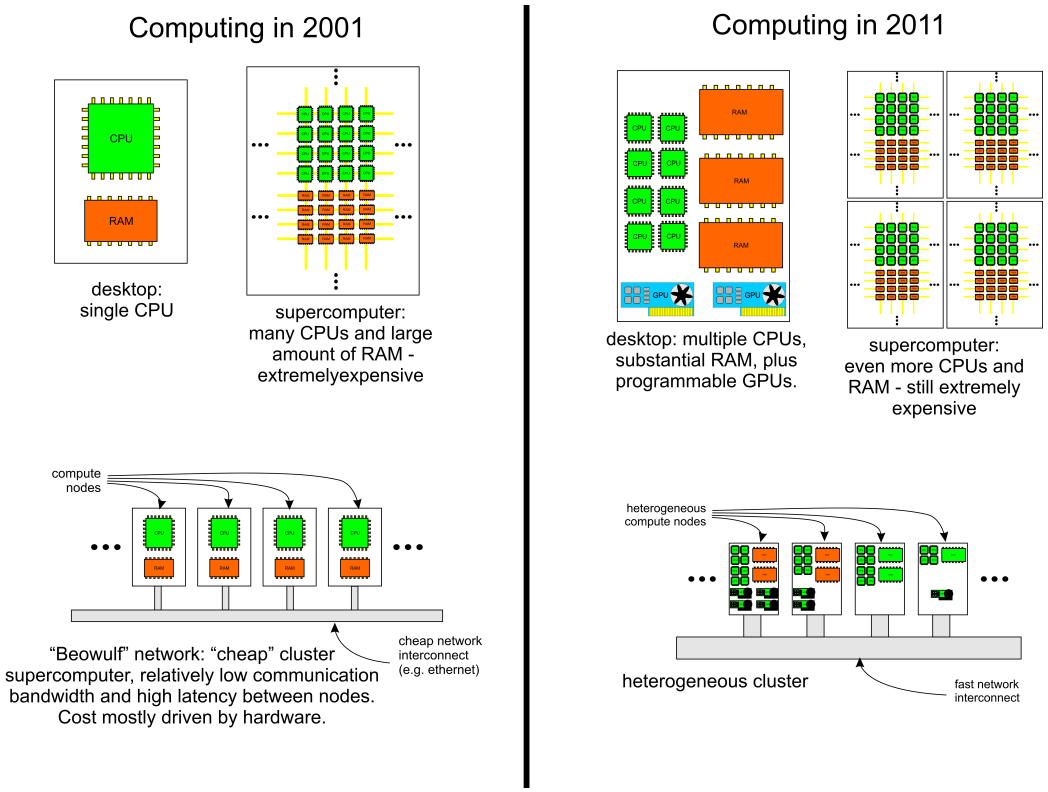


Figure 4.4: Comparison between computing architectures over the past decade: as little as 10 years ago the parallel computing landscape was relatively simple, with very expensive supercomputers offering tightly coupled shared memory resources (favouring thread-based parallelism) vs. networked clusters of cheaper individual "desktop-like" computers (favouring message-passing parallelism). Parallel computing on the desktop did not exist. Today the situation is far more heterogeneous, with desktop users having processing resources that resemble small shared-memory supercomputers, and supercomputers offering mixed CPU/GPU architectures.

4.4.2 Parallel Architectures

There have been several methods developed to parallelise programs across multiple compute-nodes, and to a large extent these are architecture-dependent. The choice of parallelisation paradigm often depends strongly on the parallel architecture on which the program will be run, and particularly the bandwidth and latency of the communications link between processors and between banks of memory. For example on a large shared memory computer (e.g. the Cray supercomputers of the past, or their current successors in the SGI Altix range) in which all processors can communicate with each other and memory at high speed/low latency (through highly specialised network links/topologies), a thread-based parallelisation paradigm implemented using OpenMP [142] can allow the programmer to most effectively exploit the parallel resources. However shared memory computers of this type are generally very expensive, so a more common (cost-effective) parallel architecture is the networked cluster of independent compute-nodes, each of which has the same computing power as the average desktop computer. The total number of processors and RAM can be comparable to that of the shared memory supercomputer, but each processor can only access its local RAM and any inter-processor communication must be done via the relatively slow network. Under this scenario a message-passing parallelisation paradigm such as MPI makes more sense, and the algorithm must be carefully designed to take into account the latency of inter-processor communications (i.e. avoid communications at all cost and/or ensure that the messages can be passed "in the background" whilst each processor continues to do useful work).

Historically desktop computers have comprised only a single processor and a limited amount of RAM so parallel programming for the desktop has not been a major concern. However modern computers have multi-core CPUs containing as many as 10 processors: they are beginning to resemble small shared-memory supercomputers, so software aimed at desktop users may require parallelisation to maximise use of resources and increase throughput, using similar methods to those originally designed for supercomputers. Further complicating the situation is the emergence of General Purpose Graphics Processing Unit programming (GPGPU programming), in which graphics cards can be used as an additional processing resource. Graphics cards contain a very large number of "simple" floating point processing units[3], whose existence is largely due to the graphics card market being driven by consumers' desire to play games animated in real-time. These graphics-processors can be harnessed by the programmer, and also resembles a shared-memory supercomputer residing on the graphics card within the desktop computer (with certain limitations on the type of program which

[3] simple in this case meaning: a reduction in data or instruction caching compared to the multi-level caching typical in CPU architectures; clocked relatively slowly; highly optimized for single-precision arithmetic, rather than double precision.

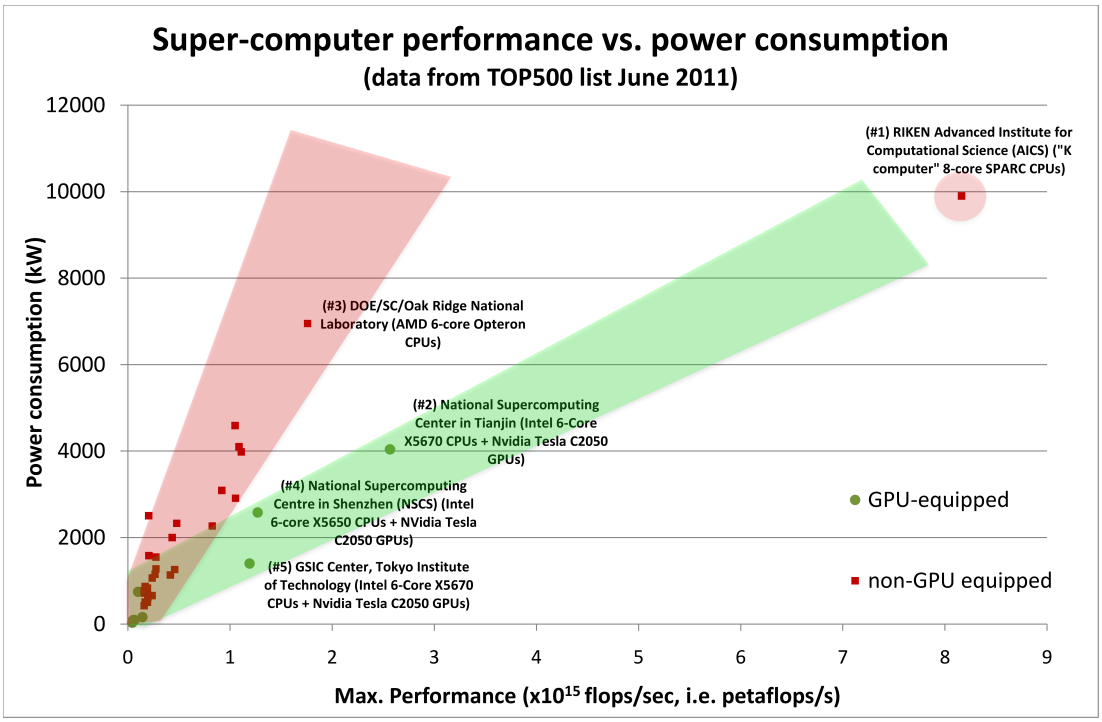


Figure 4.5: Performance vs. power consumption for supercomputers in the TOP500 list: GPU-equipped supercomputers in general exhibit lower power consumption per flop (falling within the the green-shaded envelope on the graph) than CPU-only supercomputers (red shaded region). However at the time of writing the fastest supercomputer does *not* follow this "trend".

can be run).

The increased parallel power of desktop computers has a knock-on effect on the supercomputing landscape: since many supercomputer clusters are built from desktop-like compute-nodes, each node of a cluster is now generally a multi-core node, perhaps with one of more GPU cards, leading to a heterogeneous computing environment in which a combination of thread-based parallelism and message-passing may be the optimal parallel strategy. The uncertainty of computer architecture makes it difficult to write parallel programs which work well on all computer architectures which users might employ. The shift in the parallel computing landscape is illustrated in Figure 4.4.

Attempting to predict future trends in computing has always been a difficult task, however it is possible to make certain general observations with some confidence. Figure 4.5 plots the performance of the most powerful supercomputers (in quadrillions of floating-point operations per second (petaflop/s), vs. power consumption in kilowatts) and we can see that increases in performance comes with a corresponding increase in power consumption. This is not a sustainable trend: already the most powerful supercomputers consume an amount

of power comparable to a small town[4]. It can be argued (generally by graphics card manufacturers) that GPUs are able to deliver higher performance per Watt than CPU-only architectures: the green shaded region on the graph, highlighting the general trend of GPU clusters, is less steep than the corresponding region for CPU clusters (shaded pink). From this we can conclude that the future of supercomputing is likely to be heterogeneous, so writing programs which can exploit the GPU resources as well as CPU resources is likely to be a good strategy in the long term[5].

4.5 Charm++: A parallel programming framework in C++

For this project we have chosen to use the parallel programming framework Charm++, developed and maintained by the Parallel Programming Laboratory at the University of Illinois [143, 144]. The research aims of Charm++ are stated to be:

- improve performance, while improving programmer productivity
- allow complex, irregular and dynamic applications to be developed quickly and to perform scalably on machines with thousands of processors.

In essence Charm++ provides a set of C++ classes which allow the programmer to implement parallel objects in a familiar object-oriented way. Charm++ provides an abstraction for parallel objects called *chares*. The business of running the parallel program across multiple computational nodes is handled by the framework, with the chares being distributed over the computational nodes and dynamically load-balanced according to any of various pre-defined strategies which are provided by Charm++. The fact that Charm++ abstracts the nature of the parallel computer from the programmer allows a degree of freedom in implementing a parallel program: code written and compiled using Charm++ can be run on any platform which is supported by Charm++, and that includes all of the major parallel computer architectures which the user is likely to encounter. Using Charm++ the programmer can worry slightly less about the specifics of the heterogeneous computer architectures described above, since Charm++ hides the details from the programmer, allowing them to concentrate on breaking the parallel program into chunks which are mapped onto the actual hardware by Charm++. For parts of the algorithm where the programmer knows that data

[4] Assuming a household consumes on average 1kW of power, 10MW power consumption is equivalent to about 10,000 houses.

[5] On the other hand the latest data point on Figure 4.5 (the "K computer" at AICS, Japan, with maximum performance of around 8 petaflop/s) breaks this general trend, suggesting that future CPU-only architectures may become more efficient which would reduce the benefit of GPUs.

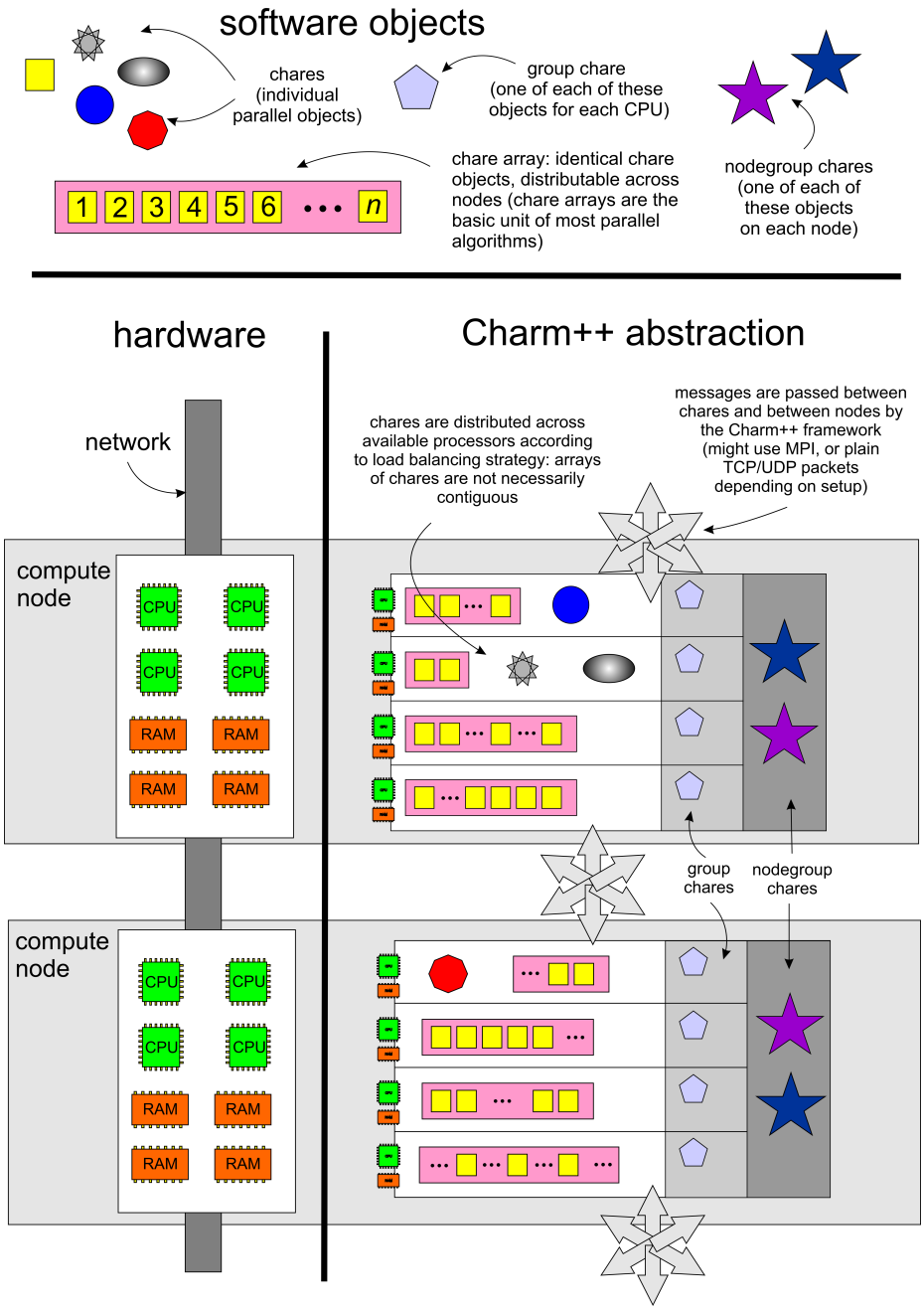


Figure 4.6: Illustration of how Charm++ chares (which are analagous to C++ objects) map onto processors/nodes. Within a parallel program chares communicate by calling entry methods (C++ functions) on each other. Communication between chares is carried out by the Charm++ framework, which serialises function parameters and transmits them across the network as messages. The exact means by which data is transferred over the network depends on how Charm++ has been compiled on the system: MPI versions of Charm++ will use the system MPI library to pass the data (which may have been optimized for a specialised network interconnect), or alternatively TCP/UDP packets can be sent directly over the network. Nodegroup chares are one-per-node, group chares are one-per-processor; all other chares (including those which are part of a chare array) reside on one processor at any given time, but can be freely migrated to any processor (by the Charm++ framework, transparently to the programmer) on any node to achieve load balancing.

packing and communication could be a bottleneck, the framework makes it possible for the programmer to exert slightly more fine-grained control, but this becomes difficult, rapidly.

In general individual chares are directly equivalent to C++ objects (i.e. instances of C++ classes). At the start of the program a mainchare object is created which spawns everything else in the program, then "entry methods" (these can be thought of as just C++ functions) are invoked on the chare objects which are executed asynchronously (rather than synchronously as in a "normal" C++ program). Charm++ manages function calls as a queue of entry methods to be called on chares. Chares are allocated to a single processor, and stay there unless migrated to a different processor by the load balancer. Since chares are allocated to a single processor, only one entry method can be executed on a chare at a time, so the programmer doesn't generally have to consider the problems of thread-safety as only a single thread will be accessing the data on a chare at a time.

The communication of data between chare objects is achieved in the usual C++ way, that is by passing data as parameters to functions: Charm++ automatically carries out any necessary serialization for the default C++ types (including STL containers), and for any other C++ classes the programmer provides a simple Pack-Unpack (or PUP) function which tells Charm++ what member data of a class needs to be packed. Since function calls are asynchronous, return values from functions need to be communicated using "Callback functions" which pass data back to the original calling chare by invoking an appropriate entry function.

The basis of most parallel algorithms is the decomposition of the problem into an array of identical chares, each chare representing one part of the total computation. Operations on chare arrays can include invoking the same function on each chare in the array, broadcasting data between chares in the array, or carrying out a parallel reduction in which some quantity is summed over all chares in the array yielding a single result.

Common data which is needed by all computational nodes is abstracted in Charm++ by the "group" chares, which are chares in the same sense as those described above (i.e. they are C++ objects, with asynchronous entry methods). Group chares behave like a global read-only variable: there is one copy of the chare on each processor: any entry method on any chare can access the "local" copy of the group chare object and access the contained data or normal C++ functions (as opposed to those designated as Charm++ entry methods) by obtaining a direct memory pointer and treating the object like any other C++ object. Data can be synchronised or reduced across group chare objects by calling entry methods on all group chares simultaneously. The idea of a group chare is important both for providing global read-access for data, and for collating intermediate results to avoid unnecessary communication overheads: each chare on a processor can deposit intermediate

results with a group chare object (writing to the group chare memory directly, rather than by passing a message), then when all chares have done this the group-level chares can perform a reduction by communicating data across processors.

The final variety of chare is the "nodegroup" which is almost the same as a "group" chare except that there is one such object per compute-node rather than per processor: on a multi-core computer the processors can all access the same physical memory, so nodegroup chare objects can be common across processors (though steps must be taken to provide synchronization in order to avoid two processors trying to write to a piece of data in memory at the same time). In a single-CPU system the concept of a nodegroup chare is identical to a group chare.

The relationship between Charm++ chares and the underlying hardware are illustrated in Figure 4.6.

4.6 Parallelising the Fast Multipole Method

Splitting the functions of BEEP into Charm++ chare objects at the top level is relatively simple: one parallel object controls each major task, as shown in Figure 4.7. However, it is clear from the profiling data in Figures 4.2 and 4.3 that in order to parallelise BEEP, the underlying FMM algorithm must be parallelised, since it is responsible for two of the three main tasks of the program (calculating the right-hand-side vector (using "vanilla" FMM) and the 12-fold BEM-FMM iterations of the GMRES loop). The remaining parts of the program are either negligible (reading in data, writing output) or more trivially parallelisable (pre-calculating the near-field BEM integrals, which are simply a large number of independent calculations: relatively easy to split across processors).

Here we present two approaches we developed for parallelising the "vanilla" FMM algorithm (the methods and results also apply with appropriate scaling to the 12-fold BEM/FMM version), and give results for how these performed when tested over a number of processors.

4.6.1 Approach 1: Fully distributed octree cubes

Our first attempt is illustrated conceptually in Figure 4.8: we split the FMM octree into cubes, and distributed the cubes across processors (as part of a larger chare array). Each "FMM Worker" chare is responsible for computing the multipole expansion within it's volume, converting the multipole expansion to plane-waves and passing them to other cubes (i.e. FMM worker chares) in the interaction list, then converting the aggregate waves col-

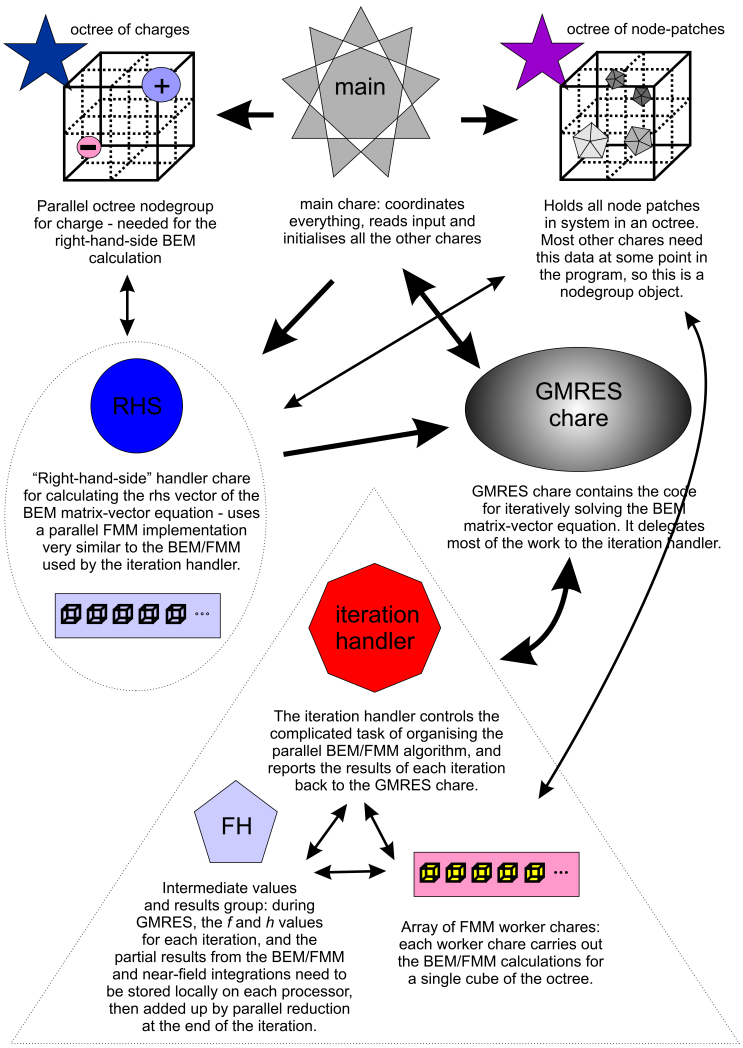


Figure 4.7: BEEPp components as Charm++ chares: the mainchare initialises and instantiates all of the other chares. The locations of node-patches and charges are stored in octree structures which are common across all processors (they are nodegroup chares). The main task of carrying out a BEM/FMM iteration is controlled by the GMRES chare and Iteration Handler chare, which spawn an array of "FMM Worker" chares, which are responsible for the actual work carried out and are by far the most numerous types of chare created by BEEPp.

lected within that cube to a local expansion[6]. Finally evaluation of the local expansions (i.e. calculating the resulting potentials and fields at each location) is carried out by whichever chare corresponds to the appropriate leaf-node of the FMM octree, as that is where the local expansion finally ends up. The same chare is responsible for the near-field summation. In order to avoid data redundancy a "master" FMM-octree exists on each compute-node as a nodegroup object.

This distributed design was intended to maximise the opportunity for load-balancing across processors. The communication between workers on the same compute-node should be very fast since local memory pointers to the data can be used (assuming an SMP memory model on each compute-node in which separate threads running on a multi-core CPU can access the same memory); on the other hand communication between chares on separate compute-nodes is expected to be slow since messages must travel across the network. In order to minimize this communication overhead the worker chares were grouped such that each compute-node was allocated a spatially contiguous set of worker chares, and messages to remote chares were aggregated on each compute-node prior to sending, minimizing the total number of messages and improving the latency-per-communication. It was intended that for sufficiently substantial problems (i.e. when the worker chares are numerous) the latencies in communicating data would be hidden "in the background" while chares were doing other useful computation. The useful work in our case would be the near-field summations (or near-field BEM integrations, in the context of the BEM/FMM).

Figure 4.9 shows the results for a test of the parallelised vanilla FMM, calculating the potential and field at each of 100,000 point charges. Although the method does become faster with increasing numbers of compute-nodes, the scaling is obviously sub-linear and gives diminishing returns for additional investment of compute-nodes. Examining the network activity as a function of time (Figure 4.10b) for the 32-processor test (run on the "non-GPU" cluster at Daresbury, with 8 processor cores per compute-node) it is clear that a large quantity of data is communicated during the interaction-list phase (red peaks, at around 4.5 seconds on the x-axis). Figure 4.10a, which illustrates what entry methods each processor is executing at any given time, shows that the construction of the plane-wave expansions (purple bars) and their transmission between FMM worker chares (the region occupied by red bars) is by far the dominant activity on most processors[7]. A similar pattern is evident in

[6] In addition each chare must interact hierarchically to pass multipoles upwards, and local expansions downwards, at the appropriate point in the FMM algorithm. However this is considerably less work than the plane-wave interactions.

[7] The time axis starts at about 2.35 seconds as this is the point at which the parallel program starts to do actual work in the main chare of the program: the previous 2.35 seconds is the time taken for the Charm++ to instantiate and synchronise 32 processes across 4 compute-nodes and so is not relevant for the purposes of evaluating the parallel performance. In all reported results we have subtracted this Charm++ startup overhead.

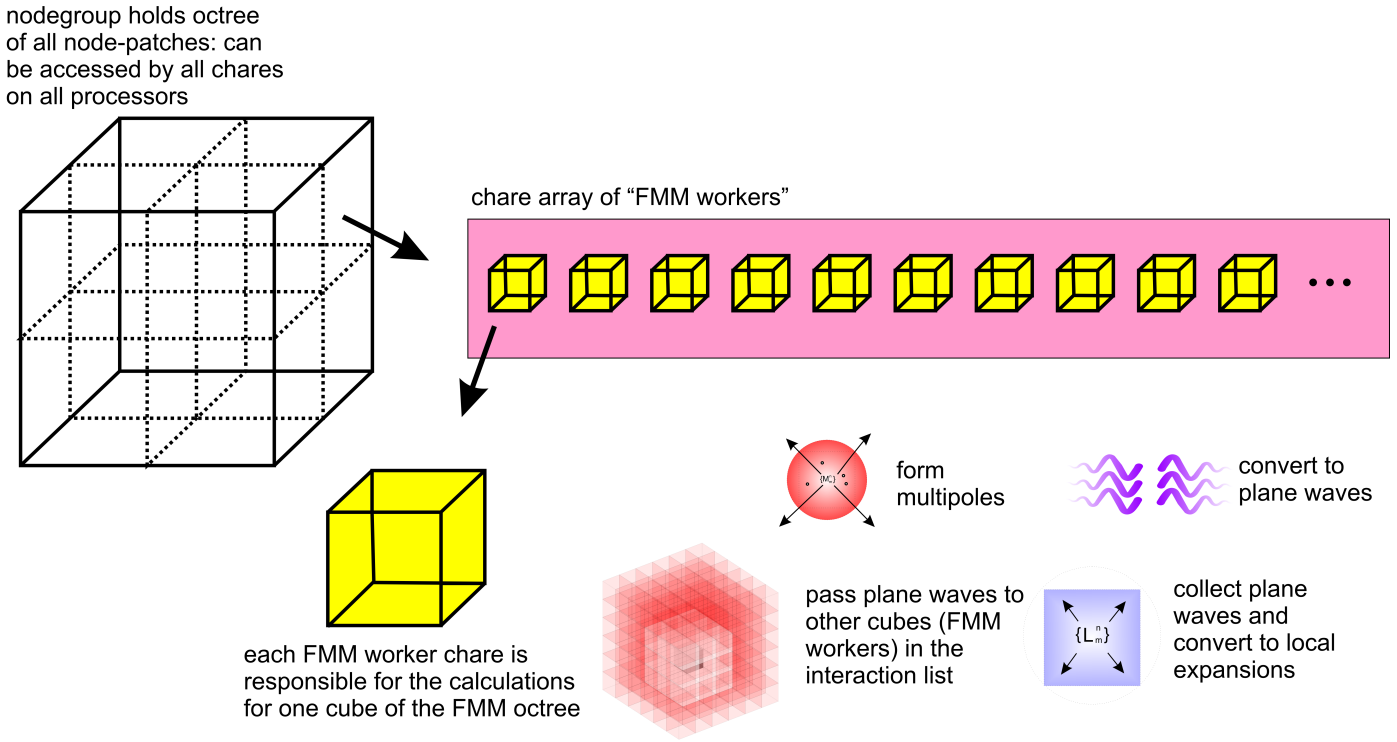


Figure 4.8: “FMM Worker” chare arrays are the basic unit of work in BEEPp: each chare in the array is responsible for calculations associate with one cube of the entire octree. Since there are a large number of cubes in the octree, there will be a large number of individual chares in the arrays, which can be evenly spread over the relatively small number of processors, leading to hopefully good load balancing. On the downside, each FMM worker chare must communicate plane-waves with a large number of other FMM worker chares, which may not be local (i.e. since worker chares are evenly spread it is likely that many chares in any given interaction list will reside on a different processor) leading to a large amount of communication.

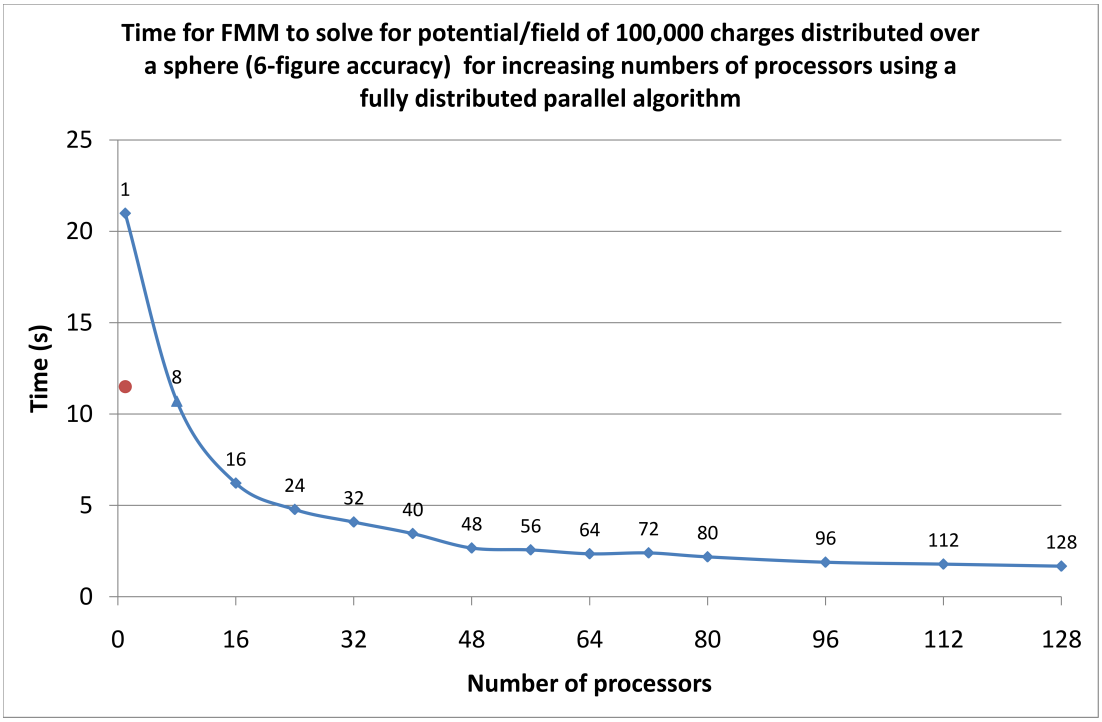


Figure 4.9: "Fully distributed" parallel FMM algorithm: time taken for evaluation of potential and field (to 6-digit accuracy) for 100,000 point charges on a spherical surface (approximating the geometry of problem relevant for the BEM/FMM in BEEP) as a function of number of processors (8 processors per compute-node). The algorithm scales somewhat sub-linearly for a relatively small number of processors, and with diminishing rates of return when adding successive compute-nodes. Compounding this poor performance is the fact that 8 processors on a single compute-node barely outperform the original single-threaded C++ code (red circle). (These measurements were taken on the "non-GPU" cluster at Daresbury; see Section 4.2.1)

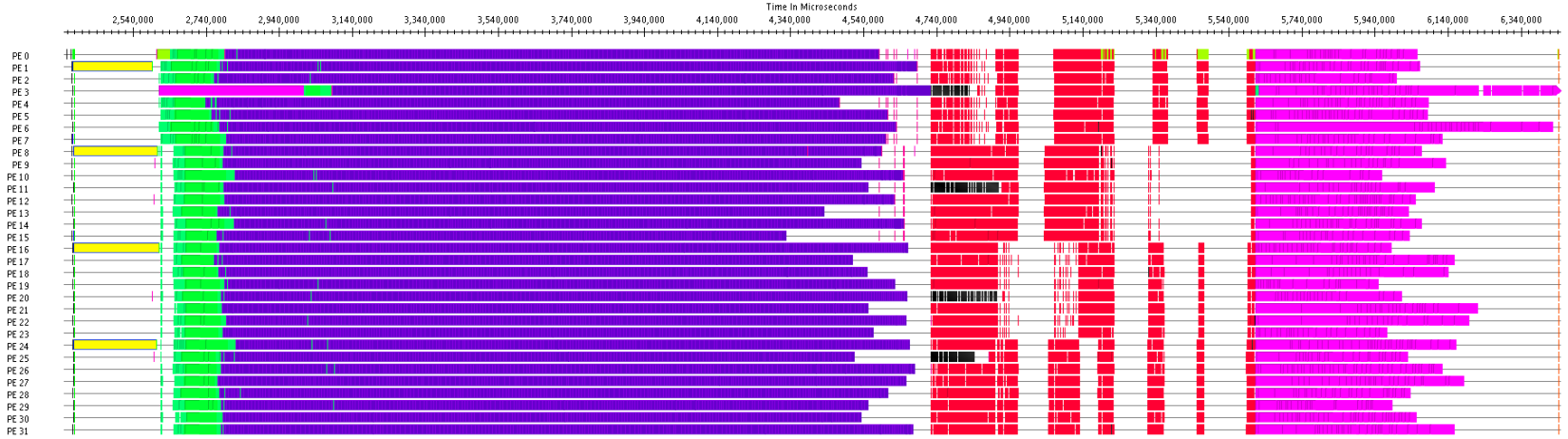
the timelines for the 16 processor (2 compute-node) case (Figure 4.11a). The construction of the plane-waves is a purely local activity involving manipulation of data in local memory, so the limiting factor here is either internal memory bandwidth or cache performance (or both). The transmission and receipt of aggregated plane-waves is limited by the available network bandwidth, and the time taken to process each inbound message.

In this case we are using an Infiniband network with a theoretical throughput of about 4GB/s, and the total size of the plane-wave data is a little over 0.6GB which is concentrated by Charm++ into relatively large 1MB chunks before transmission. Since the time taken to receive all incoming plane waves (red bars; spanning about 0.85 seconds) is considerably longer than the expected time to transmit this quantity of data (around 0.15 seconds), this would suggest that the messages are experiencing some bottleneck effect (at least a few tenths of a second, corresponding to the white "idle" gaps between the red bars). The bottleneck is less severe in the case of 16 processors (Figure 4.11b).

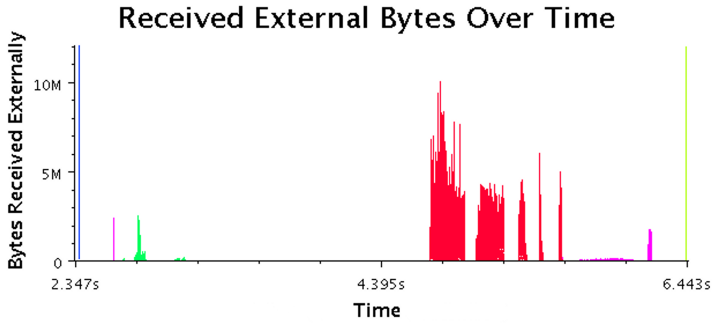
Unfortunately without lower-level profiling information than Charm++ offers it is not possible to say whether the bottleneck is due to the network hardware/software layers or occurs within Charm++ itself, for example where Charm++ breaks the incoming messages out of the large 1MB chunks into individual messages for each chare, and enqueues them for processing). In either case no simple remedy exists since we have already gone to considerable lengths to minimise the communications overhead.

Comparing the two timelines, for 32 processors and 16 processors, it seems clear that the time spent in the downward pass (including communication of plane wave data) is roughly proportional to the number of compute nodes being used, so regardless of communication bottlenecks, the sublinearity in performance must be arising elsewhere. In contrast the construction of the plane waves (purple bars on the timelines) does not appear to scale at all linearly between 16 and 32 processors, indicating that this process is not so well parallelised as we had hoped.

It is disappointing that this parallel method requires at least 8 processors to outperform the original single-threaded C++ code, denoted by the red circle on Figure 4.9 (i.e. the C++ FMM used in the non-parallel BEEP). Using the cache-simulating tool "cachegrind" (part of the Valgrind set of software development tools [145]) we can show that the number of memory accesses carried out by the parallel program running on a single processor is significantly greater than that for the single-threaded code (see Figure 4.12). This is due to the overhead incurred by aggregating the plane-wave data in thread-local memory, rather than being able to directly write the plane-wave results into the relevant shared memory. This performance penalty is unavoidable in the fully distributed scheme because the final memory address may not reside on the same compute-node as the processor generating the

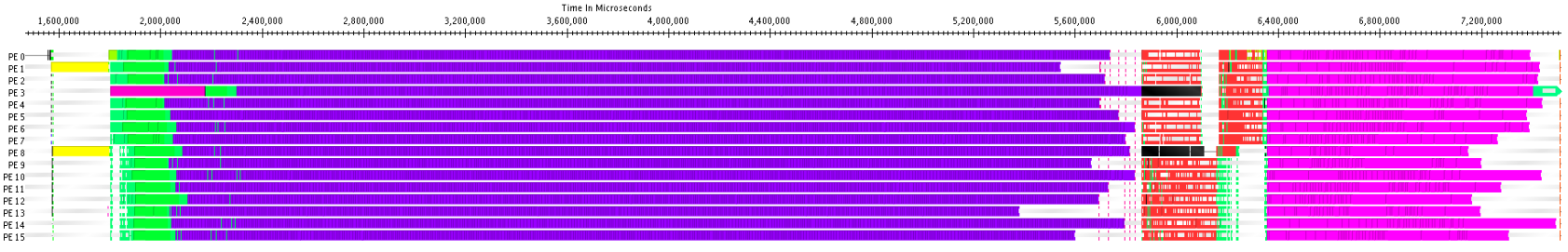


(a) Timeline of entry methods executed on each processor: creation of parallel global octrees (yellow); upward pass (green); creation of plane-waves and spatial translations (purple); collection of aggregated plane-waves at each cube and remainder of downward pass (red); evaluation of local expansions and near-field summation (pink). Idle time on each processor is indicated by the white gaps between coloured bars. The black bars are times where the CPU was not available to Charm++ (i.e. the operating system scheduled some other process to run at those times: presumably this is related to management of the network traffic generated at this point).

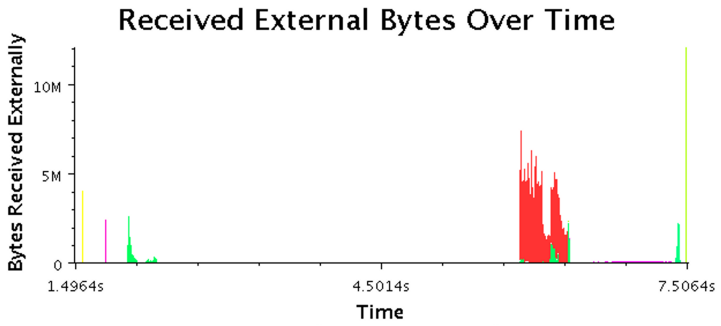


(b) Network traffic vs. time: colours correspond to the entry methods in Figure 4.10a. The red bars correspond to a total of about 0.6GB of plane-wave data being transmitted during the downward pass of the FMM.

Figure 4.10: Details for 32-processor "fully distributed" parallel algorithm: timeline of entry methods and network traffic



(a) Timeline of entry methods executed on each processor (colours as for Figure 4.10a): creation of parallel global octrees (yellow); upward pass (green); creation of plane-waves and spatial translations (purple); collection of aggregated plane-waves at each cube and remainder of downward pass (red); evaluation of local expansions and near-field summation (pink). Idle time on each processor is indicated by the white gaps between coloured bars. The black bars are times where the CPU was not available to Charm++ (i.e. the operating system scheduled some other process to run at those times: presumably this is related to management of the network traffic generated at this point).



(b) Network traffic vs. time: colours correspond to the entry methods in Figure 4.11a. The red bars correspond to a total of about 340MB of plane-wave data being transmitted during the downward pass of the FMM.

Figure 4.11: Details for 16-processor "fully distributed" parallel algorithm: timeline of entry methods and network traffic

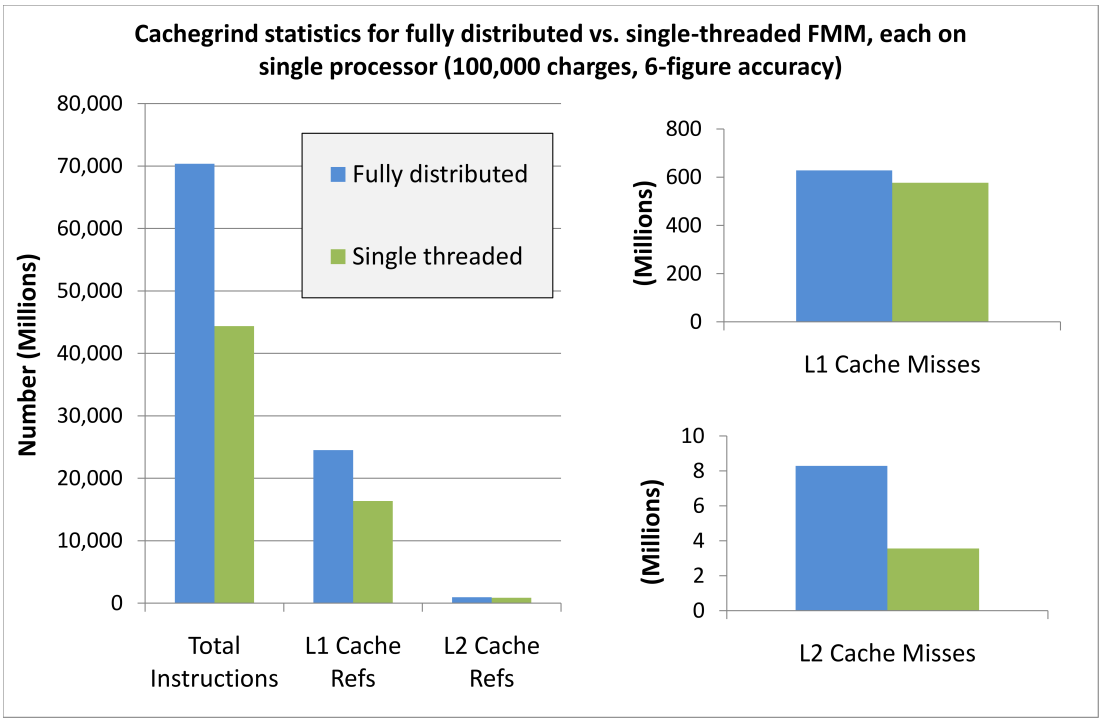


Figure 4.12: Cache statistics for single-threaded FMM vs. fully distributed FMM each running on a single processor core (but having substantially different code paths and memory access patterns). The total number of instructions fetched is much greater for the fully distributed code, and the total number of data fetches (L1 cache refs) is also increased. Although the number of L1 cache misses is approximately equal, the number of L2 cache misses (which result in a fetch from main memory, which is comparatively slow) is almost doubled for the fully-distributed code.

plane-wave data[8]. The absolute number of level 2 cache misses is also more than doubled in the fully distributed code, suggesting that overall cache-usage is slightly worse compared to the single-threaded code.

4.6.2 Approach 2: OpenMP & Redundant octrees

In order to achieve a parallel algorithm which scales across processors yet maintaining the relatively cache-friendly nature of the single-threaded code, we implemented an alternative parallel strategy which uses a more tightly thread-based parallelism (implemented through OpenMP[9]) in combination with the distributed parallelism offered by Charm++. Since

[8]Even if the processor does happen to reside on the same physical machine as the final memory address, it is not straightforward to write directly to that memory as other parallel processors may be attempting to do the same thing. We tred to improve the fully-distributed algorithm in this way, using mutexes/locks to ensure only one thread writes to memory at a time, but the overhead of the mutexes/locks was substantial (threads ended up waiting for each other a lot), and the resultant performance was actually worse.

[9]Note on confusing nomenclature: OpenMP should not be confused with MPI or the open-source implementation of MPI called OpenMPI. MPI is a message passing protocol for communication between processes on separate machines across a network; OpenMP is an open-source thread-model which allows programmers

Charm++ on our clusters use MPI for communication, this is effectively a hybrid OpenMP / MPI strategy: some processors on each node are allocated to control the parallel process at a "process" level, whilst the remaining processors on each node are dedicated to running parallel threads within each process.

Firstly we added OpenMP directives to our single-threaded FMM code in order that certain critical loops in the code exploit multiple CPUs (on a single compute-node)[10]. The addition of OpenMP directives immediately make those parts of the program multi-threaded and gave an immediate speed-up by a factor of 2 (for the quad-core office workstation described in Section 4.2.1). However OpenMP alone cannot extend the parallelism across multiple compute-nodes: in order to span multiple compute-nodes we replicated the FMM octree on each compute-node, and then solved each FMM octree independently. If each compute-node solves the entire problem using the OpenMP threaded code then clearly this is of no benefit as every processor does exactly the same work simultaneously. However, suppose we cull the FMM octree on each compute-node, such that the compute-node only concerns itself with a single cube "X" of the tree, and all of the child cubes of that cube. Then that compute-node can discard a large chunk of the remainder of the tree since it only needs to retain the data extending to the neighbourhood of the cube X (which by inheritance includes the neighbourhoods of all the children of X). Taken over all compute-nodes, all of the cubes X and their children account for the whole tree, but each compute-node only has a part of the total problem.

On the positive side this strategy minimises the communication between processes, so there should be no communications bottleneck. The actual performance is illustrated in Figure 4.13: the algorithm outperforms the original single-threaded code, and can scale across processors. The scaling is not perfect, and the overhead of initializing the octree on each processor rapidly becomes a limiting factor, but it is a significant advance over the fully distributed approach.

There are several negative features of this strategy: firstly there is no load balancing, and if one compute-node is responsible for significantly more of the tree than the rest, the total execution time will be limited by that heavily loaded processor. Secondly the strategy is inherently inefficient in respect of floating point instructions: each processor must hold not just the data for cube 'X' but for the whole neighbourhood of 'X'. Since we

to easily exploit multi-core processors in their code, simply by adding lines like: "#pragma omp parallel for" (which will parallelise a for-loop in C/C++).

[10] Assuming your compiler supports OpenMP (which most modern compilers do), adding OpenMP directives is simply a matter of adding an extra line or so of code around certain loops, so is in some cases very simple to carry out. However it is unfortunately quite easy to actually slow down the code rather than speed it up, and the programmer must understand the code very well in order to avoid various multi-threading pitfalls such as race conditions and multiple threads overwriting the same piece of data.

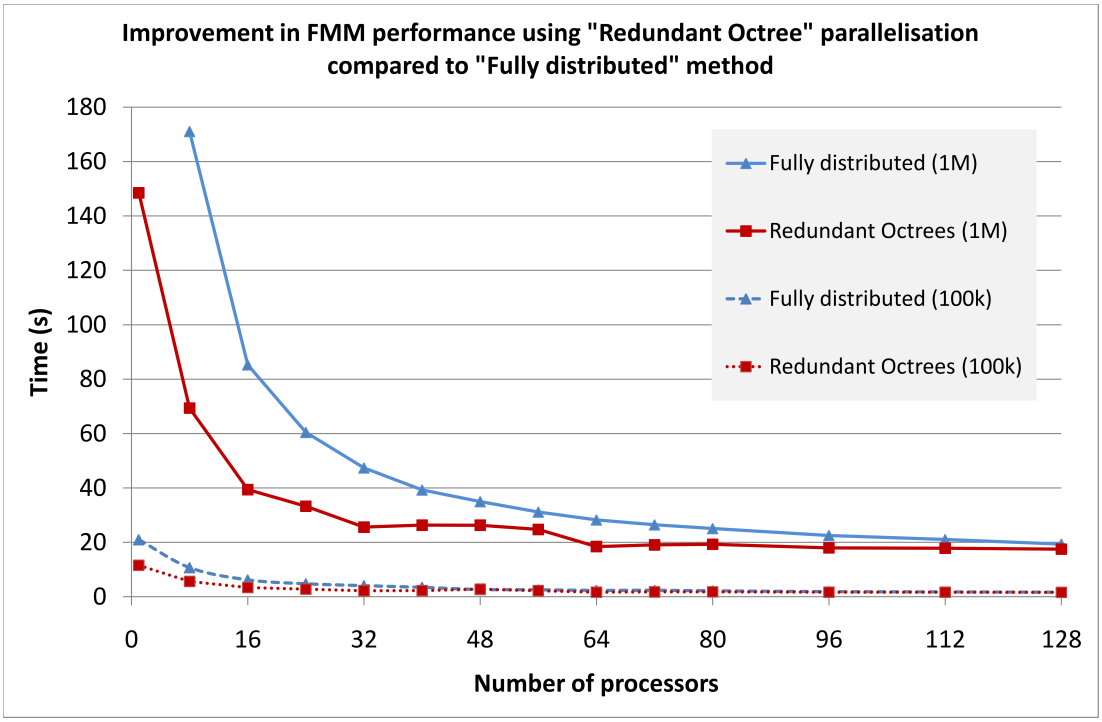


Figure 4.13: Scaling of the "redundant octree" parallel FMM for the same 100,000 point test as above, and also for a larger 1,000,000 point-charge problem, with results for the fully distributed method included for comparison. The larger problem size makes it easier to see the difference in performance between the two parallel methods, because the amount of time spent doing computation work is large compared to the overheads of the parallel program. These measurements were made using the "non-GPU" cluster at Daresbury: each compute-node has 8 processors, so for the redundant octrees method one FMM octree is created and solved on each node, using 8 OpenMP threads on each compute-node.

do not communicate across processors the multipole expansions and plane waves for those neighbour cubes (and all their children!) are calculated on each processor that requires that data: this is significant over-computation.

From Figure 4.13 we can see that there is very little benefit in using 16 compute-nodes rather than 8 compute-nodes (128 processors vs. 64 processors, since these compute-nodes each contain 2 quad-core CPUs). The reason for this is the non-balanced nature of this parallel strategy, illustrated in the timelines of Figure 4.14.

In summary, using OpenMP for finely-grained multi-threading of the FMM code allowed us to achieve a reasonable CPU-usage on each compute-node without ruining the cache-friendliness of the code. Using Charm++ to distribute copies of the FMM octree across multiple nodes gave us a simple method for higher-level coarse-grained parallelism of the overall BEM/FMM problem. Despite inherent inefficiencies this strategy, somewhat surprisingly, gives the best performance.

4.7 BEEPp

The previous section discussed our efforts to parallelise the FMM algorithm which forms the computational heart of the combined BEM/FMM electrostatics algorithm. Here we show how the above results translate into performance gains for the parallel program BEEPp. Using the "redundant octree" parallel-FMM method within the BEM/FMM we tested the performance of BEEPp for a large problem (275,000 node-patches, as described in Section 4.2.3; the neighbourhood size for the BEM/FMM octree was 1000), over moderate numbers of processors (up to 64). The results are plotted in Figure 4.15. The time taken on a single processor was 1800 seconds, compared to 170 seconds on 64 processors. The scaling relative to the absolute number of processors is admittedly sub-linear, but if we are interested in the actual time taken to solve a given problem rather than the efficiency with which we achieve that task, then BEEPp has clear advantages over it's single-threaded counterpart BEEP.

It is worth noting that running the "original" BEEP code with the OpenMP enhancements (which converts the single-threaded code into a multi-threaded code able to take advantage of multi-core CPUs) gives an immediate speedup of nearly 3x on the 8-processor machine used here. Therefore the code which will be used by the average desktop user (that is, BEEP rather than BEEPp) also benefits from this very simple parallel enhancement.

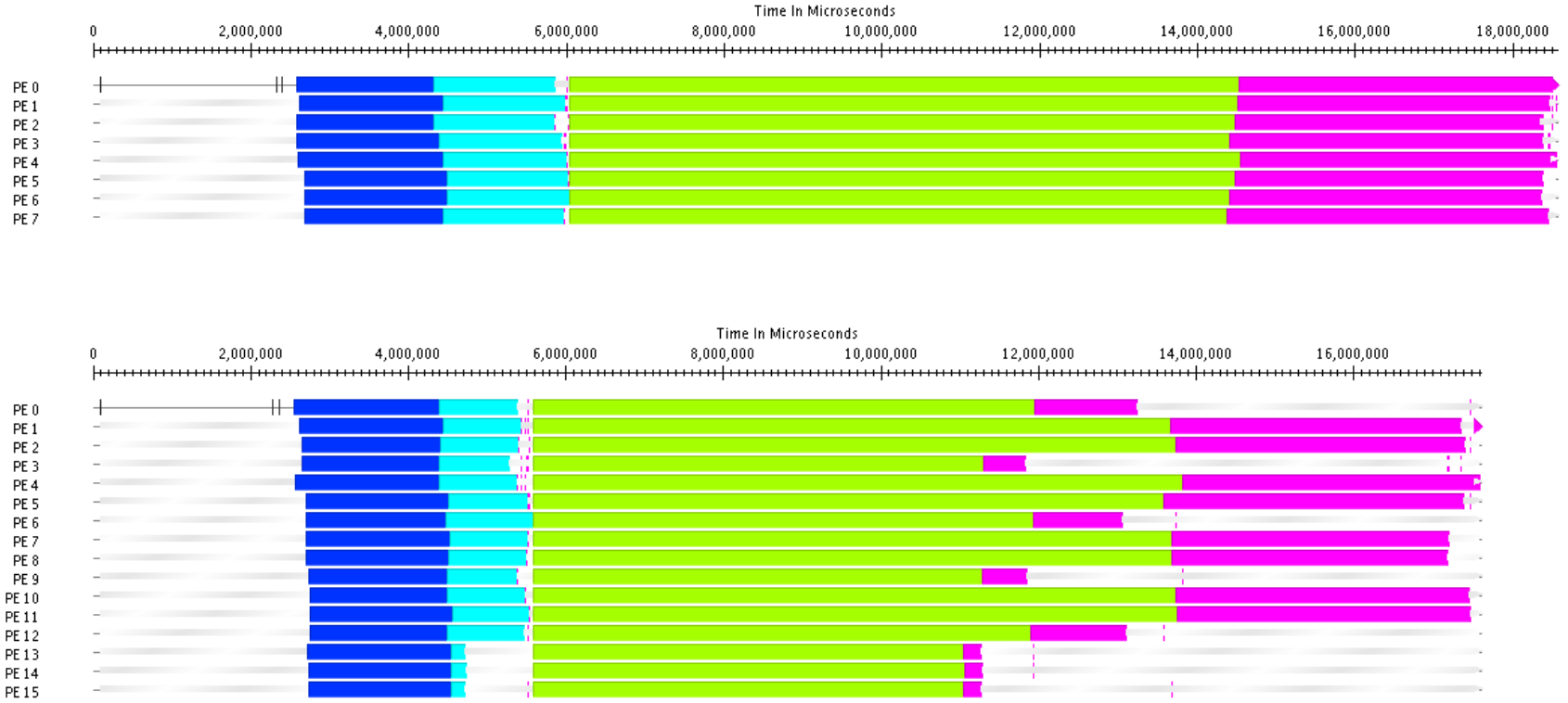


Figure 4.14: Timelines for parallelised FMM (1,000,000 point-charge problem) using 64 and 128 processors (grouped into 8-processor compute-nodes, using the "non-GPU" cluster at Daresbury, see Section 4.2.1). There is very little benefit in doubling the number of processors in this case since the parallel strategy does not allow effective load balancing, and the total time taken is limited by the most heavily loaded compute-nodes. Nonetheless, the method is still better than the fully distributed parallel method described in Section 4.6.1. Colours: creation of octree (dark blue); culling of excess nodes (light blue); total time to solve FMM (upward and downward passes) (light green); time to evaluate potentials and derivatives using local expansions and near-field summation (pink).

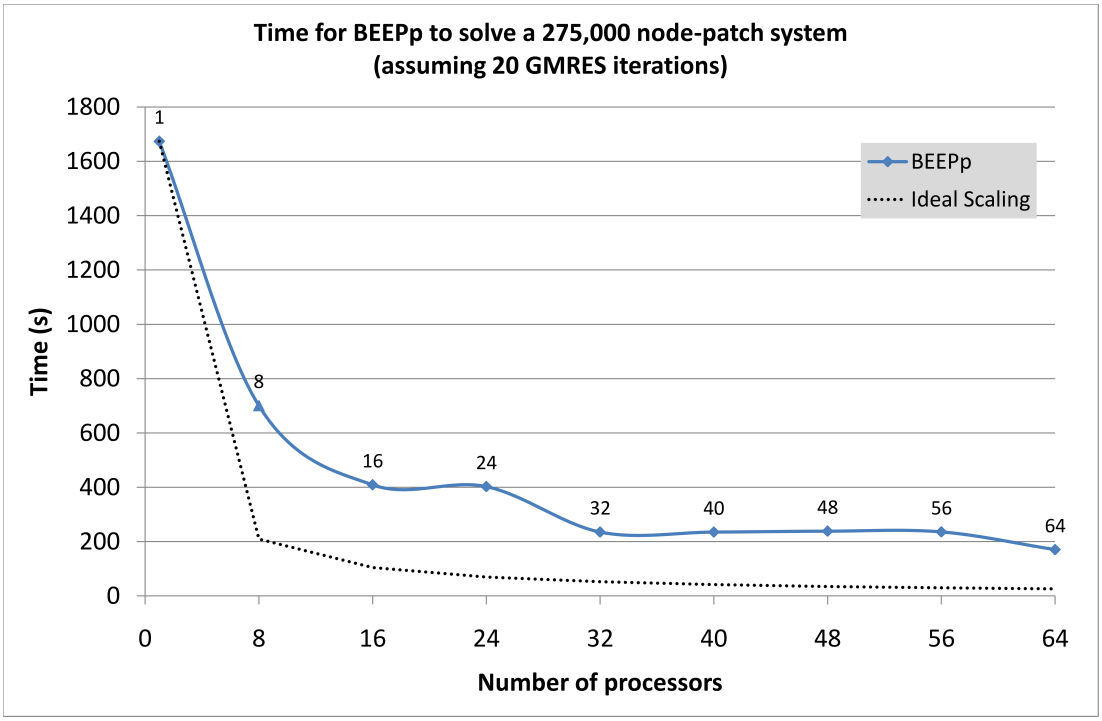
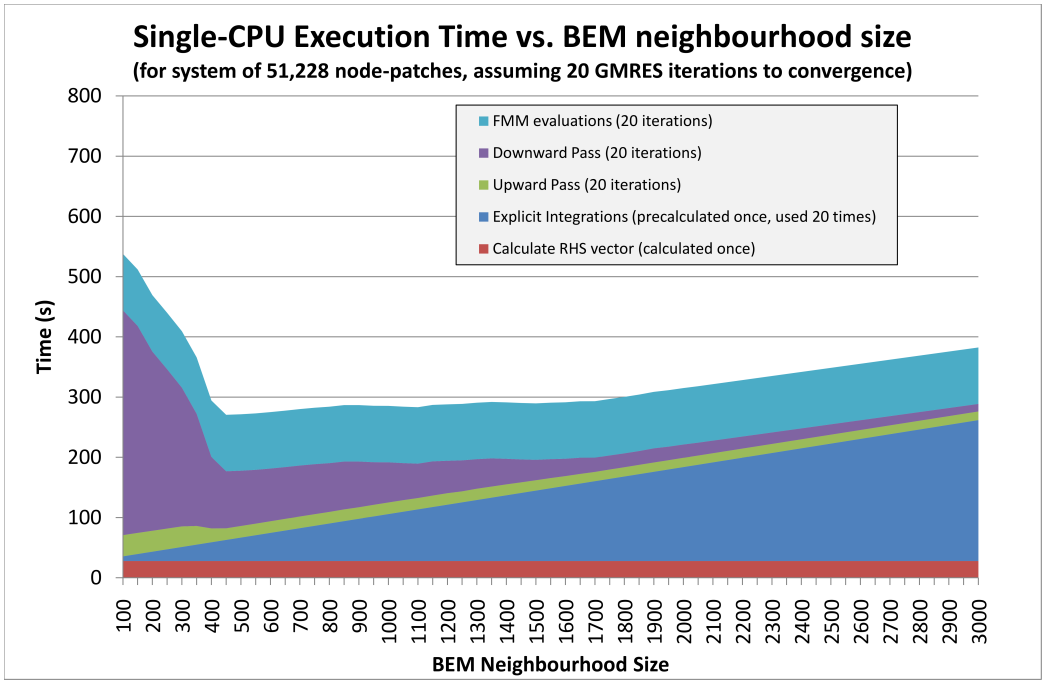


Figure 4.15: Scaling of BEEPp for a 275,000 node-patch problem (20 GMRES iterations), using up to 8 compute-nodes with 8 processors each (data collected using the "non-GPU" cluster at Daresbury; see Section 4.2.1).

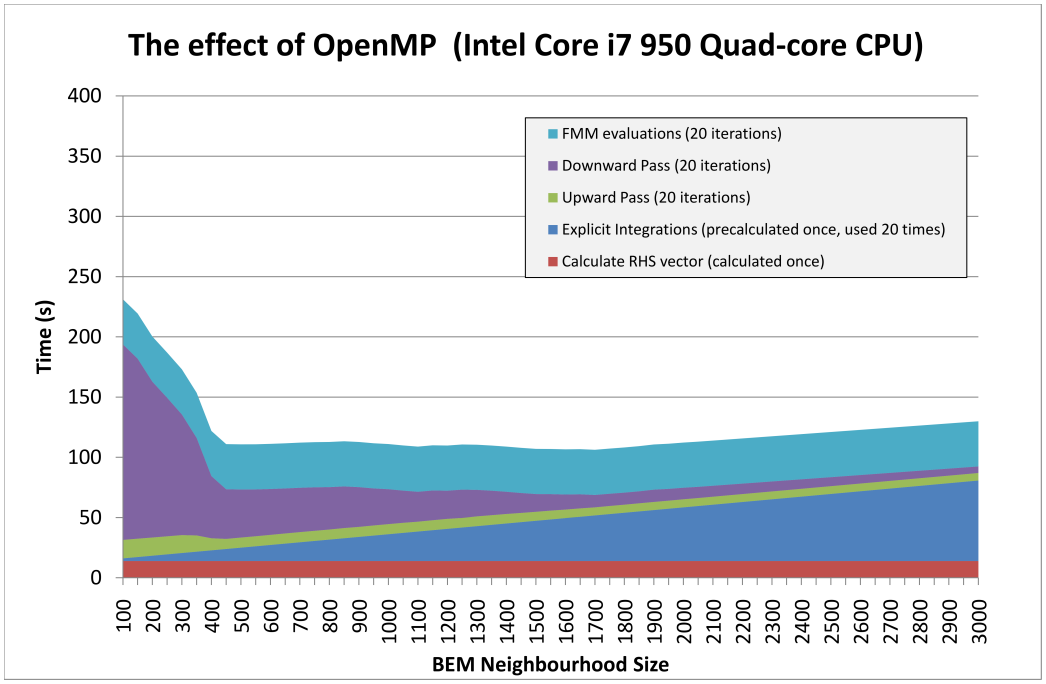
4.7.1 The effect of parallelisation on the FMM neighbourhood size

Figure 4.16a shows how the time taken to solve a system containing 51,228 node patches depends on the size of the FMM octree neighbourhood (i.e. the parameter which directly controls the number of BEM near-field integrals), and demonstrates that there is an optimum neighbourhood size, somewhere around 500, though there is not a large difference in total execution time for values between 500 and 1000. This optimum, which is controlled on one side by the level of refinement in the FMM octree, and on the other by the cost of pre-calculating near-field integrals, depends on how many GMRES iterations are expected to be required: a large number of GMRES iterations makes it worth spending extra time at the start pre-calculating near-field terms. From experience we have found that perfect spherical meshes tend to converge in fewer than 10 iterations, whilst more irregular meshes (e.g. proteins) generally require more iterations to solve the BEM (some examples of the number of GMRES iterations required for real proteins is given in Table 5.2 of Chapter 5).

When we use OpenMP to parallelise BEEP running on a workstation equipped with a quad-core Intel i7 CPU (described in Section 4.2.1) we measured a decrease in time taken to compute the near-field integrals by a factor of 3.8. The decrease in execution time for the FMM parts was somewhat less, approximately a factor of 2.0 which corresponds to our



(a) BEEP execution time for single processor core. There is an optimum neighbourhood size in this case (single-threaded BEEP, no GPU) of around 500.



(b) BEEP execution time using OpenMP and 4 CPU cores. The near-field explicit integrations are accelerated by a factor nearly equal to the number of CPU cores, whilst the FMM is accelerated by a factor of approximately 2.0. The optimum neighbourhood in this case becomes flatter: we suggest a neighbourhood size of 1000 is appropriate for multi-core systems.

Figure 4.16: Total BEEP execution time (colours indicating time spent on each task) for a "realistic" system of 51,228 node patches, as a function of neighbourhood size. The meshed system is outlined in Section 4.2.3; the measurements were taken on a workstation equipped with a quad-core Intel i7 950 CPU (see Section 4.2.1).

previous experience for the "vanilla" FMM. Both speedups decrease the execution time of BEEP markedly, giving an overall speedup of around 3x. (Figure 4.16b).

The difference in relative speedup also makes precalculations a little "cheaper" compared to FMM operations. The result is to lower the slope of the dark blue triangular region in Figure 4.16a, which corresponds to pre-calculations of near-field integrals, shifting the optimum neighbourhood size to the right.

For multi-core systems (which are the norm) we recommend a neighbourhood size of anywhere between 1000 and 1500: the "optimum" of Figure 4.16b is actually quite flat, the performance is not greatly affected by slightly non-optimal neighbourhood sizes.

4.8 GPU acceleration

As we mentioned in the short introduction to parallel computing (Section 4.4), current supercomputers (and the general-purpose HPC clusters more readily-accessible to the average researcher) are beginning to exhibit a degree of heterogeneity in hardware, featuring multi-core CPUs and compute-nodes with one or more GPU cards offering additional computing power. The recent development of GPGPU paradigms (e.g. CUDA from NVIDIA [146], the Stream SDK from AMD/ATI [147], and the open-source standard OpenCL [148]) has led to a growing community of GPU programming enthusiasts, and much money is being invested by the manufacturers to promote the use of GPU programming techniques. Whether the current trend for GPU acceleration continues in the medium to long term future is uncertain, but since at the time of writing GPU hardware is likely to be available to users of BEEP we found it worthwhile examining the possible benefits of GPUs on the BEM/FMM algorithm.

In order to make our code as portable as possible, we used the OpenCL standard which allows our code to run on either AMD or NVIDIA GPU hardware.

4.8.1 Accelerating the BEM near-field integrations using GPGPU programming

Due to the difficulty of implementing complex number arithmetic on the GPU (and the factors of 2-10 slower double precision compared to single precision floating-point arithmetic on current GPUs), we did not implement any of the FMM itself on the GPU, but instead focused on the near-field numerical integrations, since that part of the algorithm features many independent calculations (integrations between source and target node-patches) which should be highly parallelisable, and can usefully be carried out in single precision.

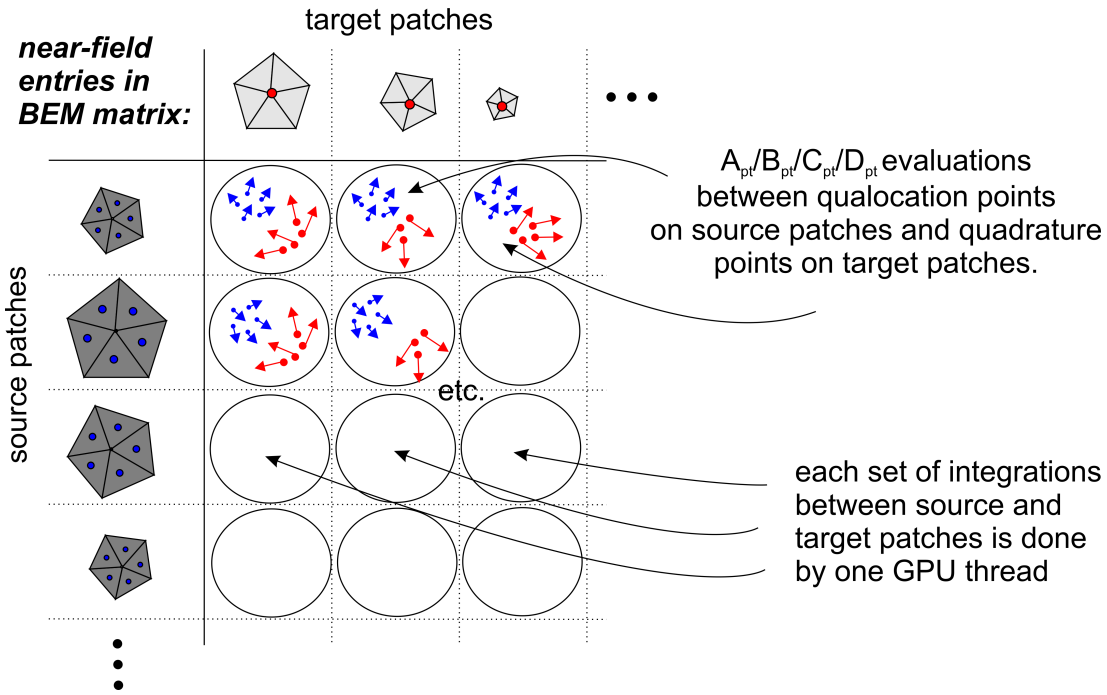


Figure 4.17: The decomposition of near-field integrations into GPU threads: each GPU thread calculates the BEM kernels for a pairwise node-patch interaction (between a node-patch in one cube of the FMM octree, and another node-patch in the neighbourhood of that cube).

The near-field integrations between node-patches were illustrated schematically in the previous chapter (Section 3.2.2.4): Figure 4.17 shows how each set of integrations (between the qualocation points on a source-patch and the quadrature points on a target-patch) are each handled by a single GPU thread. The 2-dimensional arrangement of threads in the picture correlates to the 2-dimensional set of thread-blocks which are created on the GPU: the aim is to have as many threads concurrently running on the GPU as possible, in order to maximise usage of the GPU floating-point processors, but also to mask the latency in memory accesses. The limiting factor in the number of threads which can fit within a GPU thread-block is the number of registers required by each thread to hold local data: in our case the BEM kernels are quite complicated, so require a relatively large number of registers per thread, limiting us to 4x8 threads in each block. Therefore the total set of near-field source/target node-patch interactions is divided into a number of 32-thread chunks.

The speedup of the "naive" $\mathcal{O}(N^2)$ BEM algorithm using the NVIDIA GTX580 GPU is substantial, as shown in Figure 4.18. The GPU is at least 15 times faster than the same calculations carried out on the CPU (using all available cores with OpenMP). This speedup shifts the point on Figure 4.1 at which the $\mathcal{O}(N^2)$ algorithm breaks even with the $\mathcal{O}(N)$ BEM-FMM algorithm, making it faster to evaluate small meshes (a coarsely meshed

protein, perhaps) on a single desktop machine with a GPU.

The speedup produced by the GPU depends very much on the order of numerical integration scheme chosen in the near-field integrals, i.e. the number of quadrature points chosen on the source and target node-patches, as illustrated in Figure 4.17 of the previous Chapter. If we use the more detailed Galerkin integration scheme, we can show that the GPU produces a very large speedup relative to the CPU (Figure 4.18b).

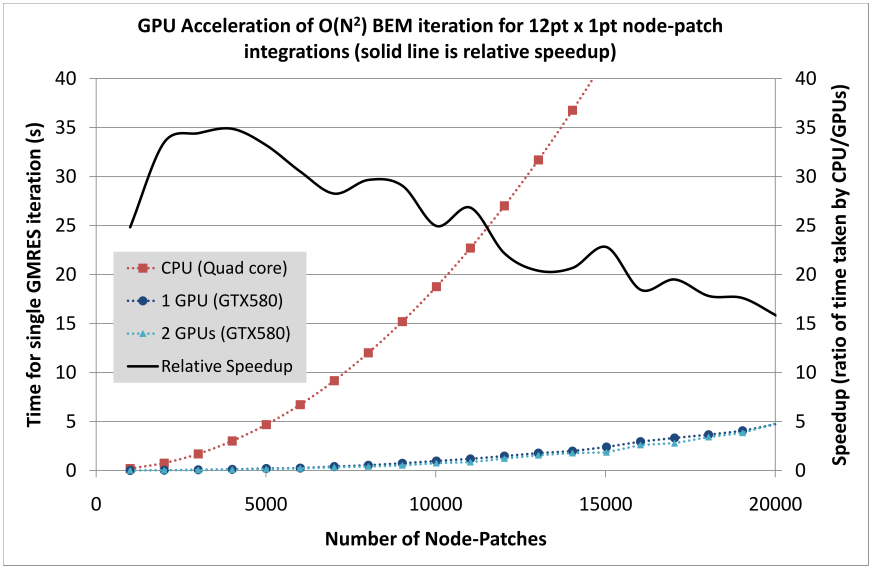
The GPU will also speed up the integrations of the near-field within the linear BEM/FMM. In fact we have two types of integrations taking place: near-field integrations between node-patches, and singular "self-patch" integrations. The former will benefit from the 15x speedup, whilst the latter will benefit even more from the 100x relative speedup.

Figure 4.18a suggests that for the less computationally intensive integration method (qualocation, corresponding to near-field integrations) we are in fact not making full use of the GPUs, as the addition of a second GPU does not produce any speedup: we are limited by the rate at which we can load data (node-patches and their quadrature points) on and off of the GPU.

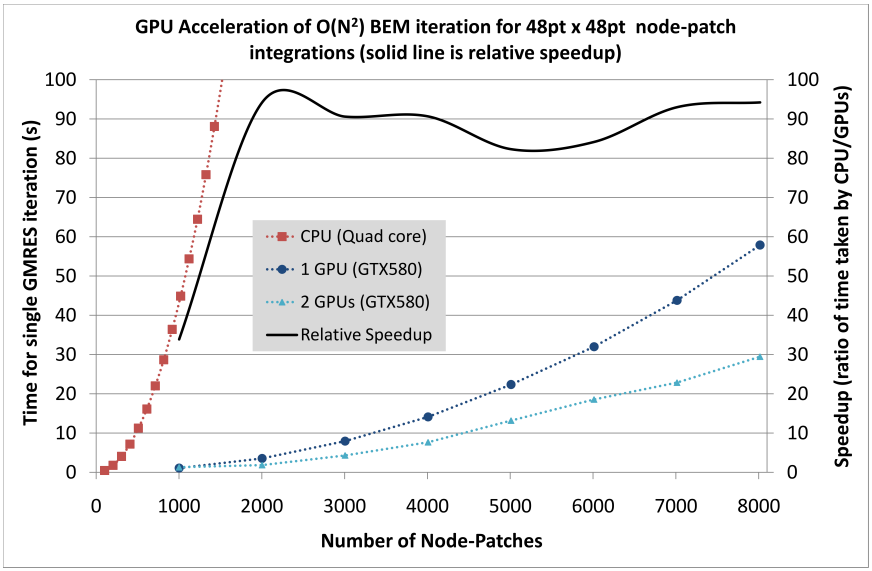
4.8.2 The effect of GPU on FMM neighbourhood size

The GPU brings a further benefit to BEEP besides simply speeding up the numerical integrations. Using a GPU to evaluate the near-field integrals, the cost of the pre-calculations is further reduced relative to the FMM procedures (Figure 4.19), which shifts the optimum neighbourhood size to an even higher value than that obtained using OpenMP. This new optimum depends on many factors:

- The speed and quantity of GPUs available, relative to the speed and number of CPU cores, taking into account possible communication bottlenecks in transferring data to/from the GPU.
- The "heaviness" of the numerical integration / discretization method (i.e. Galerkin / qualocation / centroid collocation and the number of quadrature points chosen in each case).
- The number of terms in the FMM: the time taken by the FMM in Figure 4.16a are for 9-term multipole expansions, which yields 3-digit accuracy. Fewer FMM terms would reduce the time taken for all FMM-related parts of the algorithm, at the cost of accuracy.
- Memory available to store precalculations (discussed further in Section 4.8.4)



(a) Speedup of qualocation numerical integrations: approx 12 integrations carried out per pairwise node-patch interaction.



(b) Speedup of Galerkin numerical integrations: approx 2300 integrations carried out per pairwise node-patch interaction.

Figure 4.18: Speedup of BEM numerical integrations using OpenCL (on our current-generation GPU workstation: Intel Core i7 950 @ 3.0 GHz (quad-core), with 12GB RAM and dual NVIDIA GTX580 graphics cards). These graphs show the time taken for a single GMRES iteration using the naive (non-FMM) $\mathcal{O}(N^2)$ BEM algorithm for increasing problem size (in number of node-patches). The upper figure shows (at least) a fifteen-fold performance increase for numerical integration based on qualocation (multiple integration points on source node-patch, single integration point on target node-patch). The lower figure shows a much more detailed integration scheme in which almost 200 times more numerical integration operations take place: the relative speedup in this case is even more impressive. All speedup values are relative to the same integrations carried out on the CPU (using all available cores with OpenMP).

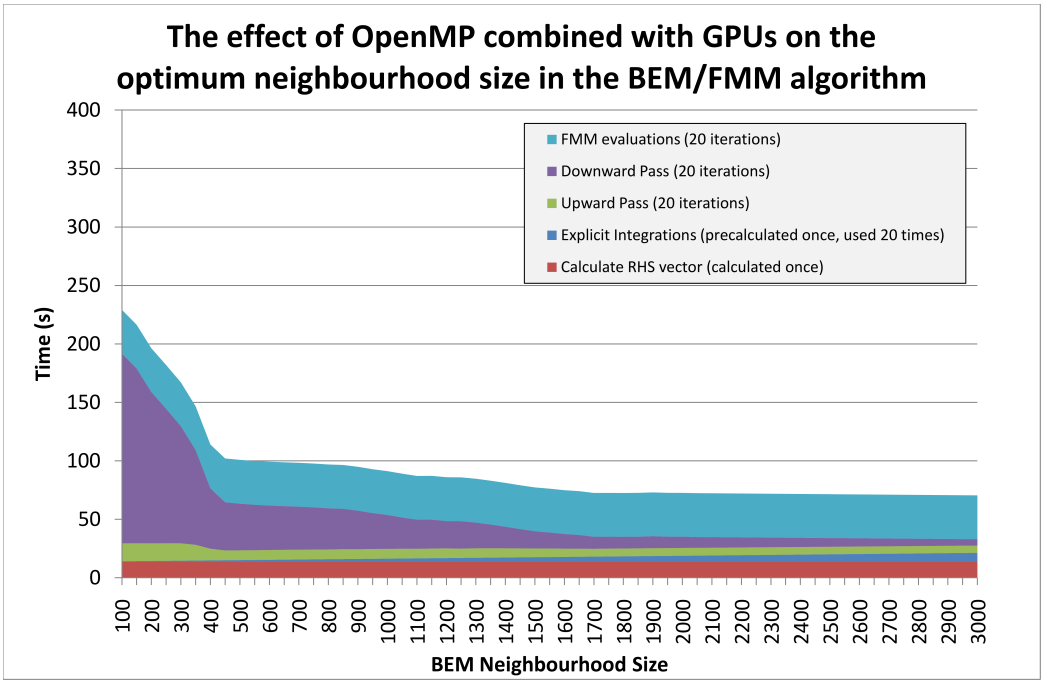


Figure 4.19: The optimum BEM/FMM neighbourhood size for shortest execution time of BEEP on a system of 51,228 node-patches.

The effect is much less pronounced when using older GPU hardware (such as the previous generation graphics cards on the Daresbury compute-nodes), where we found that there was only a very minor speedup relative to the CPUs, so anything other than the latest generation of GPUs we do not recommend altering the neighbourhood size beyond the optimum used for OpenMP (Figure 4.16b). For newer GPUs we recommend a neighbourhood size of around 2000, as settings much beyond this can lead to excessive memory usage (see Section 4.8.4).

4.8.3 The overall speedup of the BEM/FMM produced by GPU acceleration

Despite the impressive speedup of the numerical integrations illustrated in Figure 4.18, the overall effect of the use of GPUs on the linear BEM/FMM algorithm is not so dramatic because the precalculated numerical integrations do not account for more than about one quarter of the execution time. To make maximum use of the GPU, we have also accelerated the near-field terms of the "vanilla" FMM, so the calculation of the right-hand-side vector is also reduced significantly, giving a further small speedup to the overall program.

Figure 4.20 shows how GPU acceleration affects the time taken to run 20 GMRES iterations of a 51,228 node-patch system, using BEEP (with OpenMP), on a variety of computational resources, and also shows the performance of the larger 275,000 node-patch

system on a small number of compute-nodes, each equipped with 4 NVIDIA Tesla GPUs.

AMD vs. NVIDIA The AMD Opteron 2376 and Intel Xeon E5540 compute-nodes at Daresbury (see Section 4.2.1) are equipped with 3x AMD Firestream 9270 and 4x NVIDIA Tesla s1070 GPUs respectively. Both of these machines are roughly comparable in terms of numbers of CPU cores, except that the AMD Opteron has much smaller CPU cache than the Intel processors. We found that the performance of our code on the GPUs was limited by the data transfer to and from the cards, so in fact these results reflect more on the internal hardware of the compute-nodes than the GPUs themselves, which appeared from our results to be roughly comparable. The relative speedup produced by the (multiple) GPUs was small, and probably not worth the additional cost in power consumption.

Newer hardware beats previous generation The newer-generation Intel Core i7 950 CPU was slightly faster than either the AMD Opteron or the Intel Xeon, despite having only 4 CPU cores rather than 8. Most of the difference in speed is in the FMM code, which is accelerated on a single compute-node using OpenMP. It seems likely that this parallelisation strategy is unable to make full use of all available CPU cores, and is limited by memory bandwidth: this, combined with the 20% greater clock-speed, would explain the superior performance of the Intel Core i7 950 CPU over the Intel Xeon.

Our OpenCL code performs faster on the newer GTX580 card than on the Tesla, as shown by the greater relative speedup of precalculated integrals, illustrated in blue on the Figure. We have already seen in Figure 4.18a that more than one GPU does not necessarily result in additional performance gains if the problem is not compute bound: we would speculate that there is a communication bottleneck occurring between the compute-node and the externally attached server which holds the GPU cards. This is partly due to the nature of our code in which the near-field integrations are not sufficiently "compute-heavy" to fully occupy the GPUs whilst data transfers take place. The most important difference between the Tesla s1070 cards and the GTX580 is that the latter is one generation more advanced. The newer GPUs are superior for several reasons: increased memory bandwidth; increased number of registers; more relaxed memory addressing constraints; faster clocks speeds. All of these appear to make our code run very much faster on the newer hardware than the old.

Larger-scale problem gives better The relative GPU speedup of the larger 275,000 node-patch problem run on 4 compute-nodes is greater than that for a single compute-node (using the same hardware). The increase in optimum neighbourhood size leads to much reduced time spent in the FMM, but the total time spent calculating near-field integrals

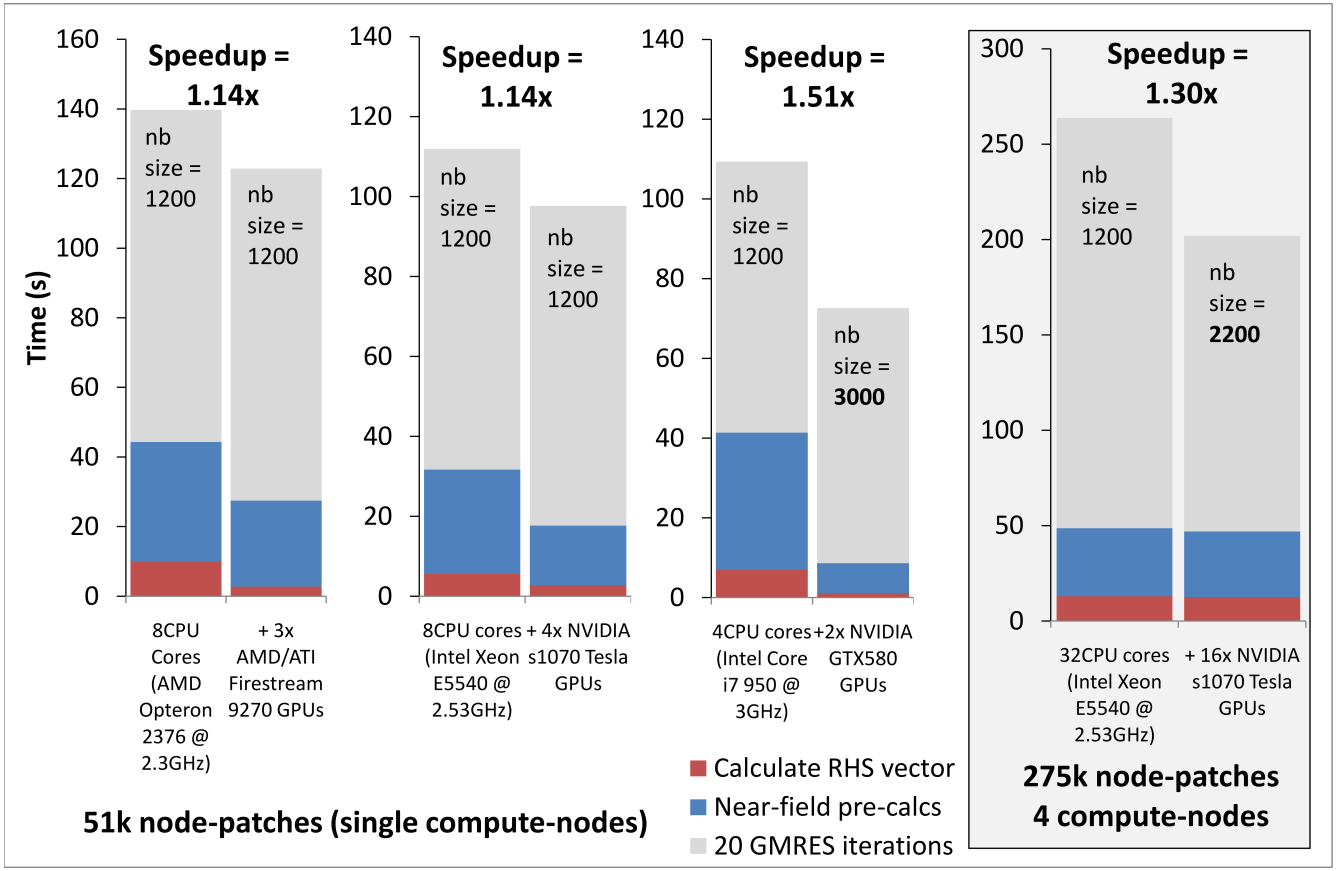


Figure 4.20: The relative speedup produced by GPU acceleration on BEEPp, for a 51,228 node-patch system and a 275,000 node-patch system. The BEM/FMM neighbourhood size ("nb size") was manually tuned to near-optimal in each case. The relative speedup is at most 1.5x, which is slightly more than would be predicted from Amdahl's Law (due to the effect of increased neighbourhood size, and due to the additional speedup of the calculation of the RHS vector).

remains the same (there are more of them in total to be carried out). The GPU has more effect in this case than for a single compute-node because the optimum neighbourhood size is larger than for the single compute-node: the increased number of nodes gives in total an increased communication bandwidth between CPUs and GPUs.

4.8.4 Maximum problem size is constrained by RAM

BEEPp should, in principle, allow almost any problem size across multiple nodes[11], though we have seen that the time taken to solve the problem may not scale very favourably with the number of nodes. On a single compute-node (e.g. a desktop compute or workstation) the maximum problem size which BEEP can tackle is limited by the RAM required to hold the pre-calculated near-field integrals, and to carry out the 12-fold BEM/FMM iterations, the most costly part of which (in terms of memory) is converting the multipole expansions to plane-waves for each level of the octree. The former memory requirement is easily predictable: it is $(24 \times nb_{size} \times N)$ bytes, where nb_{size} is the neighbourhood size and N is the number of node-patches. The latter memory requirement depends on the number of octree cubes in the system, which is strongly on the geometry of the problem and is therefore less easily predictable.

In the context of our 51,228 node-patch system (representing two molecules of acetylcholinesterase adjacent in space), the increase in neighbourhood size permitted by the GPU results in a proportionate increase in memory for precalculated integrals. With neighbourhood size of 2,000 instead of 1,000 this means the total memory used for pre-calculations goes from 1144 MB to 2288 MB. On the other hand the amount of memory required by the FMM should decrease slightly, as there are fewer FMM octree cubes, so fewer multipoles/plane-waves to be stored at any time. The peak memory usage for the test above (Section 4.8.3) actually changed from 1750 MB to 2700 MB: this is an increase of 950 MB, which means we saved about 200 MB in the FMM.

These results suggest that the maximum size of problem which can be handled by a GPU workstation with 12GB of RAM is around 220,000 node-patches (using optimal neighbourhood size of 2000), whereas without GPU acceleration the maximum problem size is around

[11] Since a global octree of all node-patches in the problem is stored on every compute-node at initialization, the actual limit is controlled by the number of node-patches which fit in the memory of the compute-node with least RAM. However since node-patch objects are relatively small (about 512 Bytes), this limit is likely to be at least a few million node-patches. If it is really necessary to solve a system with very large numbers of node-patches, regardless of time taken, then the fully-distributed parallel method discussed earlier in the chapter (as opposed to the redundant octree method) avoids the memory constraint, at the cost of worse overall performance.

340,000 node-patches (the latter allowing 7.8GB of memory for the precalculated near-field integrations and the remainder for the FMM, with a neighbourhood size of 1000) [12].

There is a way to reduce this memory constraint further: the GPU is fast enough at evaluating the near-field integrals to dispense with the pre-calculations altogether, and to carry out these integrations at each GMRES iteration, running concurrently with the CPU whilst the latter handles the BEM/FMM part. The optimum neighbourhood size now becomes the size at which the GPU and the CPU both take exactly the same time to complete the near-field integrations and the BEM/FMM respectively (otherwise either the CPU must wait for the GPU or vice-versa). Since the amount of data being transferred off of the GPU is much reduced, this strategy actually results in relatively faster operation of the GPU part of the code[13].

By removing the memory burden of precalculated near-field integrals, the maximum problem size on a GPU-equipped workstation with 12GB of RAM increases to around 1.5 million node-patches. However this is achieved at the expense of much reduced algorithmic efficiency in terms of the total number of floating-point operations executed.

4.9 Conclusions

We began this chapter by showing that although the BEM/FMM algorithm is linear, it is too slow for simulation of large biomolecular systems. We have demonstrated two methods for parallelising the underlying FMM algorithm: neither method scales very well across multiple compute-nodes, but they both allow larger simulations to be attempted than would be possible on a single workstation. OpenMP was found to be highly effective for exploiting multi-core CPUs on a single compute-node, whilst the overall effectiveness of Charm++ was limited by the communication latencies of messages between parallel objects, despite running across one of the fastest network interconnects currently available (Infiniband). Nonetheless our implementation of BEEPp can solve a protein electrostatics problem in reduced time by running on multiple compute-nodes, with 32 processors being a good trade-off between scaling and relative performance.

[12] Since we cannot exactly predict the peak memory-usage of the FMM for the general case these are estimates of the size of system we can accommodate without resorting to swapping data from RAM to virtual memory (i.e. the hard disk of the computer), based on the actual memory usage for the 50,000 node-patch system. Other processes running on the compute-node (such as operating system, GUI) will also consume some of the available memory, so the actual maximum system may be slightly more or less than these figures.

[13] The GPU code for calculating integrals "on-the-fly" is also slightly more efficient (in terms of number of hardware registers required) than that for calculating the near-field matrix terms individually, so the GPU kernel runs a little faster, in addition to the saving in memory transfers across the (relatively slow) PCIe bus.

GPU acceleration is shown to give dramatic speedups to certain parts of the BEM/FMM algorithm, however (as could be predicted by Amdahl's Law) the final speedup of the total program is more modest (1.5x speedup when using current-generation hardware), as there are significant portions of the program which cannot, at present, benefit from GPU acceleration. The presence of the GPU shifts the optimum FMM neighbourhood size, which aside from improving the relative speedup slightly, results in significantly increased memory usage which unfortunately lowers the maximum problem size possible on a single GPU-equipped workstation compared to the same machine without the GPU.

Optionally, the GPU also allows *larger* problem sizes to be tackled on a single workstation by removing the need for precalculated near-field integration results to be stored in memory: they can be evaluated on-the-fly at each iteration and discarded. This mode of operation is inherently inefficient with regards to floating point instructions, but comes with no actual performance penalty thanks to the concurrency of CPU and GPU operation, and allows for very large problem sizes to be tackled: 1.5 million node-patches on a 12GB GPU workstation (this would take approximately half an hour to complete 20 GMRES iterations).

The execution time remains limited by the amount of time spent carrying out FMM evaluations. We conclude that the best way to further improve BEEP to exploit GPU acceleration would be to move parts of the FMM algorithm onto the GPU (starting with the final-stage FMM step in which local expansions are evaluated into potentials and higher derivatives, since this is the rate-limiting step in the algorithm at this time). Our implementation of the BEM near-field integration in OpenCL runs on both AMD and NVIDIA hardware with similar performance on each. However the development of the OpenCL standard appears to be taking place at a slower rate than the advances in NVIDIA's proprietary CUDA GPGPU framework: consequently it may be simpler and easier to implement the FMM algorithm using CUDA rather than OpenCL, at the cost of portability.

The parallel version of BEEP is, to our knowledge, the only fully parallel implementation of the Boundary Element Method for protein electrostatics which takes into account salt effects and allows different dielectric constants within each meshed protein, and represents a significant advance forward in the field of protein electrostatics. We have made significant progress in enabling large-scale systems to be solved in parallel, however the compute time required to solve such systems is substantial. We suggest that for large systems of proteins minimising the number of node-patches in the problem, by meshing proteins only as much as is strictly necessary, is the only practicable approach toward simulation in the near-term. A discussion of the minimum meshing requirements for proteins is included in the next chapter, which discusses the application of BEEP to protein electrostatics and protein-protein interactions.

The underlying parallel implementation of the Fast Multipole Method (described by Greengard and Huang [112]and implemented originally in non-parallel Fortran) is also to our knowledge the only parallel implementation of that algorithm currently available. The FMM code is available as a stand-alone modular piece of standard C++ code (for operation on a single compute-node or workstation) or as a stand-alone Charm++ application, both with and without OpenMP enhancement for multi-core CPUs.

Our source code is included on the CD attached to this thesis.

Chapter 5

Protein Electrostatics and Protein-Protein interactions

5.1 Outline of this chapter

The previous chapters have introduced the implicit solvation model and the boundary element method (BEM) (combined with the fast multipole method (FMM)) for solving the linearised Poisson-Boltzmann Equation (PBE). Chapters 3 and 4 gave details of our implementation (BEEP) and its parallelisation using multiple CPUs and GPU acceleration. Here we use BEEP to model the electrostatics of proteins.

Firstly, we compare BEEP to other implicit solvent electrostatics programs and show that BEEP gives results which are consistent with other methods, giving us confidence that BEEP works correctly beyond the spherical test cases we used in Chapter 3.

Secondly, we explore the relationship between the nature of the mesh defining the protein surface and the electrostatic solvation energies calculated by BEEP. We show that using an alternative curved "PNG1" triangle representation of the original planar-triangle surface appears to improve the "accuracy" of BEM results (i.e. degree of correspondence in the value of solvation energy between the high and low detail meshes).

Thirdly, we demonstrate that we can obtain converged values for the total solvation energy of systems of proteins, accurate direct evaluation of the electrostatic forces (as suggested in the literature, and generally advertised as an advantage of the BEM for proteins) is not possible even using very detailed meshing of a protein surface. This conclusion means that dynamical simulation incorporating BEM electrostatics is impracticable. However, we

show that the change in energy as two molecules move relative to one another can be found with reasonable stability, and could be used as the basis for a Monte-Carlo simulation.

5.2 BEEP: solving the linearised PBE for proteins

We ran BEEP on 51 PDB protein structures, previously used as a test set by Tjong and Zhou [81][1]. The aim was to demonstrate that BEEP could reproduce the trend in solvation energies as found by other implicit solvent electrostatics programs for more complicated systems than those discussed in Chapter 3 (i.e. for proteins, not simple spheres). No particular biological point is being made here in our choice of protein structures, we simply wanted to show that BEEP solves the linearised PBE for non-analytic systems. The programs used in this Section are listed and described in Table 5.1. Tables of data used to generate the following figures are given in Appendix C.

Program	Description
BEEP(p)	Boundary Element Electrostatics Program (parallelised): Our implementation of the BEM/FMM algorithm, as described in Chapters 2-4.
APBS	"Adaptive Poisson-Boltzmann Solver" (by Baker *et al.* [90]): A finite-difference PBE solver, can solve either the linearised or non-linearised PBE. Convergence of results depends on the grid parameters used, and values depend on the smoothing functions applied to both the dielectric boundary and protein charges. There are several other finite-difference PBE solvers available, such as UHBD or Delphi, but we chose APBS as it is more recently developed, free and easy to use.
AFMPB	"Adaptive Fast-Multipole Poisson Boltzmann" solver: another implementation of the BEM/FMM algorithm, originally written and described prior to this project by Lu *et al.* [103, 105], but not released publicly until 2010 (by which time we had mostly written BEEP, using the descriptions in the literature). AFMPB is single-threaded Fortran code, and the underlying FMM code is that supplied by Jingfang Huang [112]. Following the public release of AFMPB (under GNU Public License), we adapted the AFMPB code for evaluating the BEM kernel functions (A_{pt}, B_{pt}, C_{pt}, D_{pt}) into BEEP since these routines were faster. Otherwise BEEP is an independent implementation of the same algorithms, with improvements (multiple dielectric constants; OpenMP/Charm++/GPU parallelisation; improved numerical integration; curved surface representation (described in Section 5.3.2)).
GB	Our implementation of a canonical Generalised Born method as described by Still *et al.* [79], with modifications for curved surfaces following the method described by Zhao *et al.* [84]. We have not attempted to apply any additional fitting to the GB energies in order to match Poisson-Boltzmann results, or to include salt effects (such as described by Tjong and Zhou [81, 82]). The GB method is suitable only for evaluating the solvation energy of single molecules.

Table 5.1: Programs for calculating implicit solvent electrostatic solvation energies.

[1] Tjong and Zhou were comparing linearised PBE solvation energies at various salt concentrations to their Generalised Born implementation, and so had a useful list of 55 PDB proteins to use as a test-set. The proteins are a non-redundant set of PDB structures with less than 10% sequence identity, a resolution of 1.0Å or better, and less than 250 residues long. We discarded 4 structures which had gaps in the main protein chain (which PDB2PQR was unable to handle), leaving 51 structures.

5.2.1 Experimental Method

The PDB codes for the 51 test proteins are given in Table 5.2. For all calculations in this section the internal dielectric of the proteins was set to 2.0, the solvent dielectric was 80.0, solvent radius (for defining the dielectric boundary) was 1.5Å, and atomic charges/radii were taken from the PARSE forcefield (using PDB2PQR). The monovalent salt concentrations were zero, 150mM or 300mM ($\kappa = 0$, $\kappa = 0.127$, $\kappa = 0.180$). Initially the ionic radii for APBS were set to 1.5Å (rather than the more usual 0.95Å and 1.81Å for Na^+ and Cl^- respectively), in order that the ionic boundary match the dielectric boundary (as is the case for BEEP/AFMPB). (As discussed in Section 5.2.3 a better match between APBS and BEEP is obtained using an ionic radius of 0.8Å.)

For BEEP, AFMPB, and the GB calculations, surface meshes were created using MSMS then cleaned, refined and decimated using Meshlab, following the method described in Appendix B. The number of vertices (node-patches) was set to approximately 2.5 vertices per $\mathring{A}^2$, though after the mesh processing steps the actual value varied between 1.73 and 3.5 vertices per $\mathring{A}^2$). Apart from the parameters already mentioned (probe radii and dielectric constants) the APBS parameters were the default values produced by the PDB2PQR script (i.e. automatic values for "reasonable" grid spacings and dimensions; smoothed molecular surface; second order spline-smoothing of charges).

The beta-version of AFMPB as described by Lu et al. [105] is available from the website of the journal "Computer Physics Communications" (`http://cpc.cs.qub.ac.uk/summaries/AEGB_v1_0.html`). The code was compiled from source using gfortran and run as per the example scripts in the distribution. The mesh-type was set to "MSMS", which internally causes a number of integration cut-offs to be applied.

BEEP was set to use a high level of detail in calculating numerical integrals: a Galerkin scheme was adopted for near-field integrals, and 4 Gauss-Legendre quadrature points were used per sub-triangle of each node-patch: assuming an average of 6 triangles per vertex of the original mesh, this gives 12 sub-triangles per node-patch, totalling 48 integration points on each source and target node-patch. Self-patch integrals (which are near singular) were evaluated in the same way but with 16 Gauss-Legendre points per sub-triangle.

We used BEEP with GPU acceleration and OpenMP parallelisation, using the 3-digit FMM (i.e. 9 terms in each multipole expansion), with a neighbourhood size of 2000.

GMRES iterations to solve system The PDB proteins and the number of iterations they required for the GMRES algorithm to solve the linear system of equations to a tolerance of 10^{-6} (magnitude of residual relative to that of the right-hand-side vector) are given in Table 5.2. The mean number of iterations for the whole test-set was 15. The number

of iterations required is reasonably consistent with no obvious correlation to the size of the mesh. The more-or-less consistent number of iterations to reach convergence suggests that the BEM system of equations is well conditioned (this was the original aim of the more complicated derivative BEM formulation over the simpler non-derivative BEM formulation).

PDB Code	surface area (Å^2)	GMRES iterations to convergence	PDB Code	surface area (Å^2)	GMRES iterations to convergence	PDB Code	surface area (Å^2)	GMRES iterations to convergence
1hje	5657.2	14	2fdn	2339.9	11	1ssx	6240.4	13
1etl	2159.3	13	1c75	4499.7	13	3lzt	5011.5	16
1ejg	3836.3	18	1yk4	2553.1	15	1m1q	5289.9	16
1p9g	3325.6	11	1vbw	3438.7	16	1nwz	4987.4	17
1wy3	5913.7	16	1c7k	5036.1	14	1cex	2308.5	9
1ucs	3910.4	17	1x6z	5162.1	15	2cws	4618.4	14
2bf9	6405.3	13	1g66	3175.9	13	1j0p	6534.7	14
1f94	6879.6	15	2chh	8185.4	20	1tqg	4816	13
1vb0	5426.8	11	1g4i	3279.4	38	1x8q	2036.7	25
2erl	4987.4	11	1u2h	7268	15	1l9l	3919.9	14
1aho	6621.3	13	1tt8	3661.8	11	1tg0	3251.8	13
1iua	798.11	16	1ufy	4657.4	18	2a6z	3037.8	10
1r6j	2982.8	16	1w0n	3367.5	14	1k4i	8393.3	10
1ok0	5108.8	16	2fwh	2132.2	16	1nls	8713.1	12
1kth	734.19	14	1pq7	1868.2	19	1iqz	3397.5	14
1xmk	7971.9	18	1gqv	6115.6	13	1eb6	6484.9	13
1zzk	4039.6	15	1a6m	7721.3	14	1exr	7577.3	17

Table 5.2: PDB codes for the test set of 51 proteins, with the surface area of the meshed molecule and the number of GMRES iterations required for BEEP to solve the BEM system of equations to tolerance of 10^{-6} . The number of iterations does not appear to correlate very strongly with the size of the protein.

5.2.2 Comparison between BEEP and APBS, AFMPB, GB

Figure 5.1 shows that the solvation energies produced by BEEP are comparable to those produced by APBS. Our results are not absolutely identical to APBS, the maximum difference being 7.9%, and an average difference of 5.0%. AFMPB also gives results slightly differing from APBS, with an average deviation lower than that produced by BEEP (2.0%) but a maximum difference of 8.9%. It turns out that the AFMPB solvation energies are generally smaller in magnitude than the APBS values, whereas BEEP energies are consistently greater in magnitude.

The primary difference between AFMPB and BEEP here is the detail of numerical integration: AFMPB does not calculate the near-singular integrals over the "self patch" and uses single quadrature-point integration with a centroid collocation discretization, which is the crudest method possible. BEEP should have superior accuracy in calculating the BEM integrals, so the BEEP results should be more reliable estimates of the "correct" solvation energy for these implicit-solvent systems[2].

[2]The extent to which these values actually match the corresponding physical quantity in reality is un-

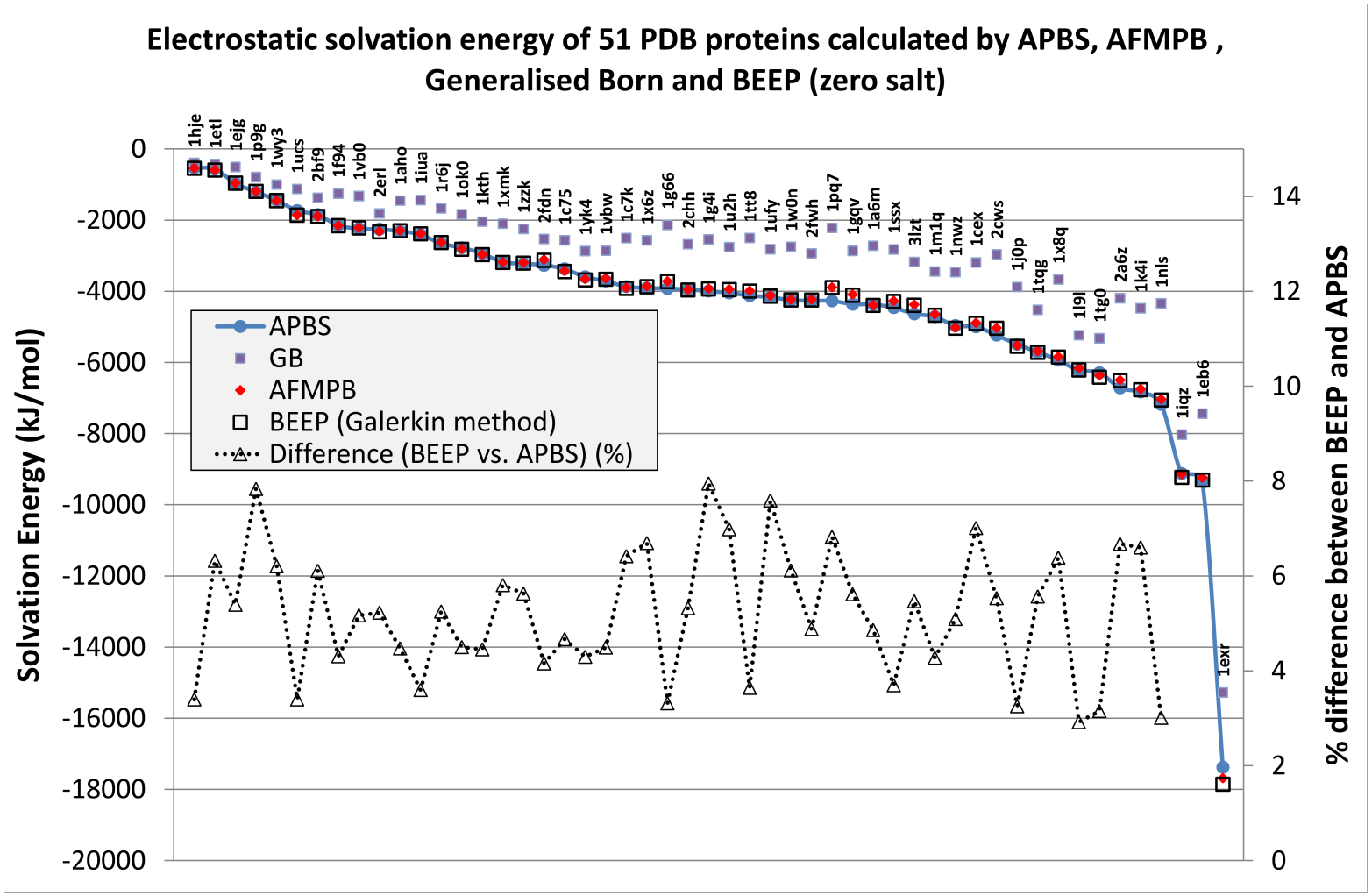


Figure 5.1: Comparison of BEEP solvation energies with those calculated by APBS (a widely used finite-difference PBE solver), AFMPB (a BEM PBE solver, with many similarities to BEEP) and Generalised Born, for 51 PDB proteins (details of the method used to calculate these data are given in the main text).

Although AFMPB gives results which are generally closer to APBS, it appears that it may be been to some extent "tuned" to do this, by use of cut-offs applied to the numerical integrations over node-patches. When the mesh type is set to "MSMS" in the AFMPB input file, integrations over nearby patches are set to zero within a 0.4Å radius, and outside of this radius the integrals are calculated by single-point quadrature. The cutoffs are present, presumably, to improve stability when using meshes directly taken from MSMS (which, as we have previously noted, can contain very slender triangles which cause difficulties for surface integration). When the mesh type is set to "OFF" (the default mesh format produced by the GAMER mesh program, which gives much "cleaner" meshes than MSMS) the cutoff is set to a value of 0.3Å, within which range integrations are carried out by a higher order quadrature rule, and outside of which range the integrations are carried out by single-point quadrature, as before. We have found that altering the cutoff for MSMS-type meshes gives significant variation in the resultant value for solvation energy. The arbitrary nature of the cutoffs in AFMPB (and their necessity for stability of the results) is clear when we convert our "clean" MSMS meshes to OFF format, and re-run them in AFMPB: the results are up to an order of magnitude greater in some cases.

The Generalised Born implementation gives substantially smaller and more variable values for solvation energy. This is broadly consistent with the findings of Feig *et al.* [83] who compared energies computed using a variety of GB methods with corresponding Poisson-Boltzmann results. The difference in results can vary enormously depending on choice of parameter set (for charges/radii) used for each method, and especially on the choice of surface (solvent accessible vs. van der Waals) since this directly affects the extent of "burial" of each atom (which is the quantity which the Born radius is approximating). In general, practitioners use an empirically derived scalings in the GB formulation to calibrate the energies against equivalent Poisson-Boltzmann results (such as described by Tjong and Zhou [81]). We have not attempted any such calibration since it would necessarily depend on the choice of parameters used for the atomic partial charges and radii, and we feel is therefore not particularly useful in the general case. Nonetheless we concede it is likely that more complicated formulations of the GB method, using variants of the canonical GB formula intended to "match" the molecular surface used by PB solvers, would be capable of achieving better correspondence with our results.

known because it is not possible to directly measure the electrostatic solvation energies: we are trying to gauge how well each program solves the mathematical system of equations describing the implicit solvent model for each protein, not reproduce any specific experimental result.

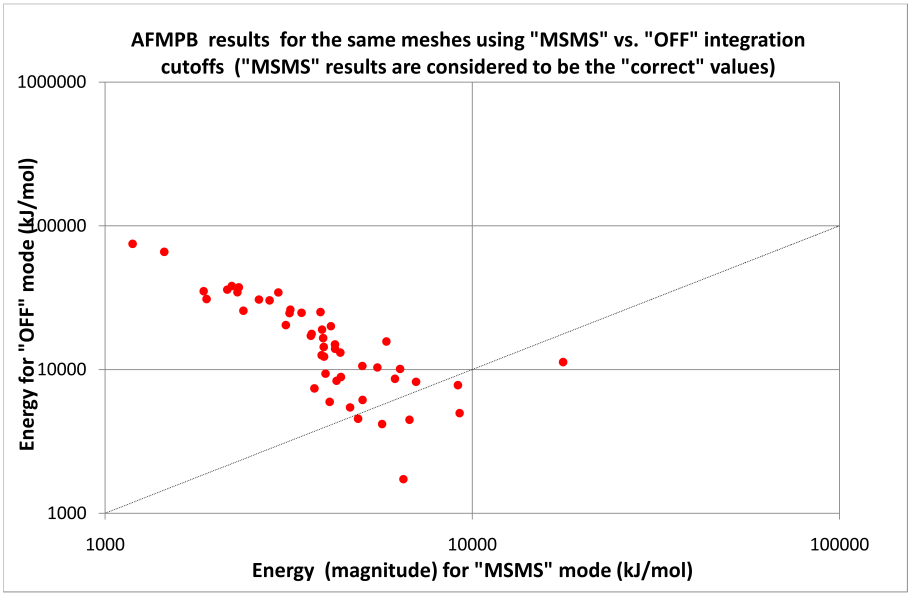


Figure 5.2: The effect of integration cutoffs on numerical stability of AFMPB: results of solvation energy for 51 PDB proteins using the "MSMS" mode (cutoffs apply within a 0.4Å radius, single-point quadrature employed beyond cutoff) and "OFF" mode (more detailed integration between 0 and 0.3Å radius, single-point quadrature thereafter). Both sets of results correspond to the *same* meshes so results should lie on the 1:1 diagonal line.

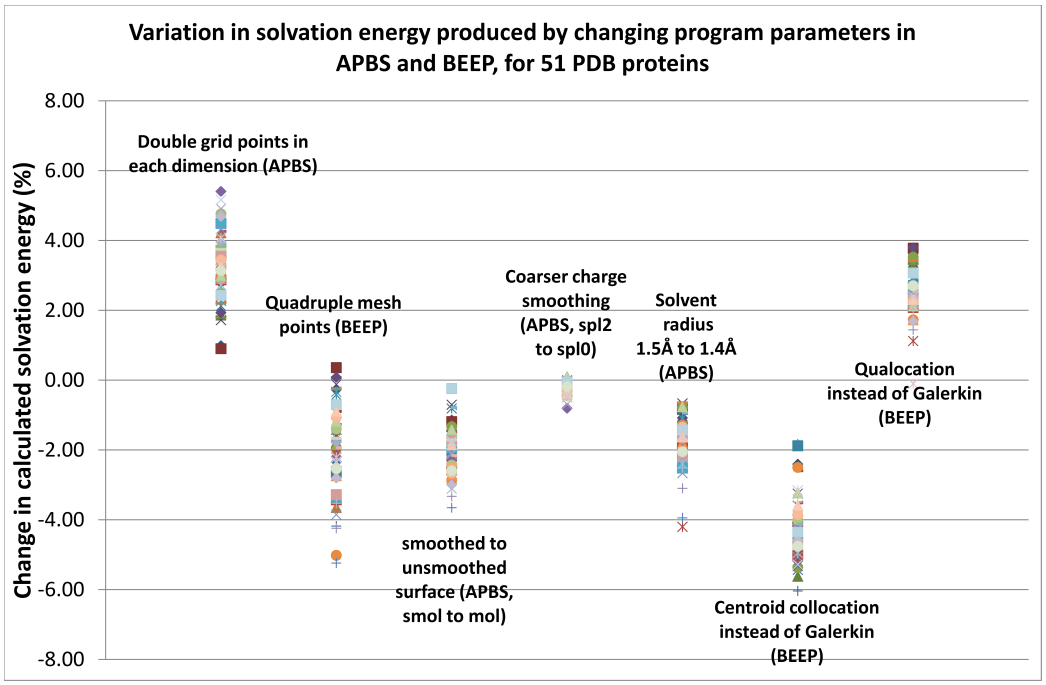


Figure 5.3: Variation in solvation energies of 51 PDB protein structures as program parameters are changed (each symbol on the plot corresponds to one of the 51 proteins). The magnitude of the percentage change is relative to the APBS or BEEP value plotted in Figure 5.1. Negative % changes are decreases in solvation energy (i.e. closer to zero), whereas positive % changes are increases in solvation energy (i.e. values become more negative).

5.2.2.1 Convergence

Figure 5.1 suggests that BEEP gives results in broad (though imprecise) agreement with APBS using "default" behaviour of both programs, but does not give any indication of the convergence (or lack thereof) in the calculated values. By increasing the number of grid points in APBS, and the number of mesh points in BEEP, we can measure how much the values of solvation energy change with an increase in the detail of the representation of the integration volume (APBS) or surface (BEEP). Figure 5.3 shows that simply increasing detail in the APBS grid can reduce the solvation energy by between 1% and 5.4%. This makes the APBS results further from the BEEP results. Additionally, the values calculated by BEEP are increased (by up to 5.2%) with increased mesh detail. The average change in energy for increased detail in APBS is -3.2%, and +2.0% for BEEP, which suggests that while BEEP values are in general slightly more well converged than the APBS results, the "true" disagreement between them is even greater than that suggested by Figure 5.1.

5.2.2.2 Surface and charge representations

APBS and BEEP differ in their representation of the protein surface (i.e. the dielectric interface) and the atomic charges, which may better explain the difference between APBS and BEEP results in Figure 5.1. The APBS solution depends quite strongly on the smoothing applied to the solvent excluded surface, which can be set to one of four values (using the "srfm" keyword in the APBS input script):

1. mol: all grid points of the total volume where a solvent molecule can be placed without overlapping a solute atom are assigned as exterior dielectric; the complement of this volume is the "internal" dielectric. (This is the usual way to define the solvent excluded volume). If the solvent radius is set to zero this would produce the volume outside of the van der Waals surface of the protein. No smoothing is applied.

2. smol: the dielectric regions are defined as for "mol" but the resultant assignment of interior/exterior dielectric constant is smoothed over neighbouring grid points by a harmonic function.

3. spl2 or spl4: a spline-based surface which, according to APBS documentation, should only be used with a parameter set of atomic radii/charges intended for spline-based surfaces of (i.e. not PARSE).

As well as surface smoothing, APBS also smooths the atomic charges whereas BEEP treats them as point charges. The placement of the charge distribution in APBS can be specified by using the "chgm" keyword, with "spl0", "spl2" and "spl4" giving increasing degrees of spline-based smoothing of the atomic point charges over sets of nearest-neighbour grid points.

Figure 5.3 shows the change in absolute solvation energy which results from changing from smoothed "smol" to unsmoothed "mol" surfaces and from a change in charge distribution from "spl2" to "spl0". The magnitude of the change in solvation energy is up to 4% for the surface parameter (in the direction of the BEEP results, i.e. more negative) and a less significant change of less than 1% for the reduction in charge smoothing.

Also of importance is the solvent probe radius used to define the dielectric boundary. Although we have used a value of 1.5Å radius in both cases, our post-processing of the low-quality MSMS surface may have produced a surface which has slightly different location[3]. If we allow the value in APBS to vary slightly, for example a 0.1Å reduction in solvent radius

[3]Obviously the mesh cleaning and refinement/decimation steps aim to minimise the total change in the surface definition, however it is not possible to simultaneously improve the mesh without changing the surface to some extent, especially in regions with high curvature or sharp geometric features. Unfortunately such regions are likely to be highly surface-exposed atoms, which have a large contribution to the solvation energy.

to 1.4Å, the solvation energy becomes more negative by up to 4% (a comparable effect to that produced by lowering the surface smoothing).

Taken together, it seems plausible that these different representations of the dielectric boundary account for a large proportion of the discrepancy between APBS and BEEP in determining the solvation of a charge distribution representing a protein in solution.

5.2.2.3 BEEP discretization/integration method

BEEP results also vary somewhat according to the numerical discretization/integration method employed (i.e. Galerkin, qualocation or centroid collocation methods, as described in Chapter 3). Figure 5.1 shows that the centroid collocation results in solutions with slightly larger magnitude solvation energy, whilst qualocation gives results which are closer to the Galerkin method, but which are generally less negative (you may recall that we saw a similar trend for the solvation energy of the Born ion in Chapter 3). These results suggest that the numerical methods of qualocation and centroid collocation provide similar precision but systematically inaccurate energies compared to the Galerkin method for real protein structures. Qualocation offers the better compromise between accuracy and performance.

5.2.2.4 BEEP solves the Poisson Equation

Taken together, Figures 5.1 to 5.3 allow us to conclude that BEEP successfully solves the Poisson equation (i.e. the PBE without salt effects) for realistic protein structures as well as for the idealised spherical systems examined in Chapter 3. The details of how APBS treats the dielectric boundary and the atomic charges has a significant effect on the output, which makes direct comparison between APBS and BEEP an "unfair test": strictly speaking they are not solving exactly the same problem. We suggest that the major algorithmic differences between the finite-difference and BEM methods is sufficient to account for the disagreement in absolute value between APBS and BEEP. For the proteins tested here, using the default APBS parameters and protein meshes produced as described in Appendix B, BEEP and APBS exhibit similar levels of convergence.

5.2.3 Changes in Solvation Energy with Salt Concentration

The absolute values of solvation energy are not in themselves particularly useful. Of more interest is the *change* in solvation energy which occurs when we perturb the system in some way, for example by changing the structure of the protein or by changing its environment.

The linearised PBE enables the changes in energy to be computed as a function of salt concentration. Ideally we would like the change in solvation energy for different salt concentrations to be consistent even when the numerical integration regime is reduced from Galerkin to qualocation or centroid collocation, because the full Galerkin treatment is computationally expensive.

Adding salt allows us to test how well BEEP solves the linearised Poisson-Boltzmann equation, compared to the same calculations by APBS.

Figure 5.4 shows the change in solvation energy for the 51 PDB proteins in our test panel as we add 150mM of monovalent salt (e.g. NaCl) to the implicit solvent model, as calculated by APBS, AFMPB and BEEP.

The correlation between APBS and BEEP is very strong (with regards to which proteins result in large changes in energy, and which do not). However, as with the zero-salt systems, BEEP results are consistently greater in magnitude than the APBS values, by 41.5% on average. The AFMPB results are generally closer to the APBS results, but have a much larger difference for some of the more highly charged systems.

In some ways it is disappointing that APBS and BEEP do not agree more closely on the change in solvation energy due to the addition of 150mM salt. Both programs are solving the same model under the same set of assumptions (linearised Poisson-Boltzmann equation) so it would be comforting if they gave more similar results. On the other hand, we have already stated in Section 5.2.2.2 that the difference in surface treatment has significant impact on the absolute solvation energy. Although we have found that changing the dielectric surface representation (from smoothed to unsmoothed) in APBS does not bring the difference in energy any closer to the BEEP value (unlike the no salt case), we have found that the ionic boundary definition does have an effect.

APBS implements a separate ionic exclusion region around the protein from the dielectric boundary, whereas in BEEP the two boundaries are necessarily identical. The default behaviour of APBS is to use an ionic probe radius equal to the maximum ion radius specified by the user: in the case of NaCl, this would be 1.81Å[4], resulting in an ionic boundary outside of the dielectric boundary. We have tried to eliminate this technical difference between APBS and BEEP by assigning ionic radii equal to the probe water radius (1.5Å). However we can coerce APBS into giving results which are much closer to BEEP by reducing the ionic radii to 0.8Å (which is somewhat below the 0.95Å ionic radius for sodium, and very much smaller than the 1.81Å radius of a chloride ion). This allows ions to approach more closely to the atomic charges (as they are represented in APBS) and thus adopt positions

[4]The choice of "correct" ionic radius is somewhat debatable, since ions are likely to be solvated: it could be argued that the ionic radius should be increased by the radius or diameter of a water molecule to account for this.

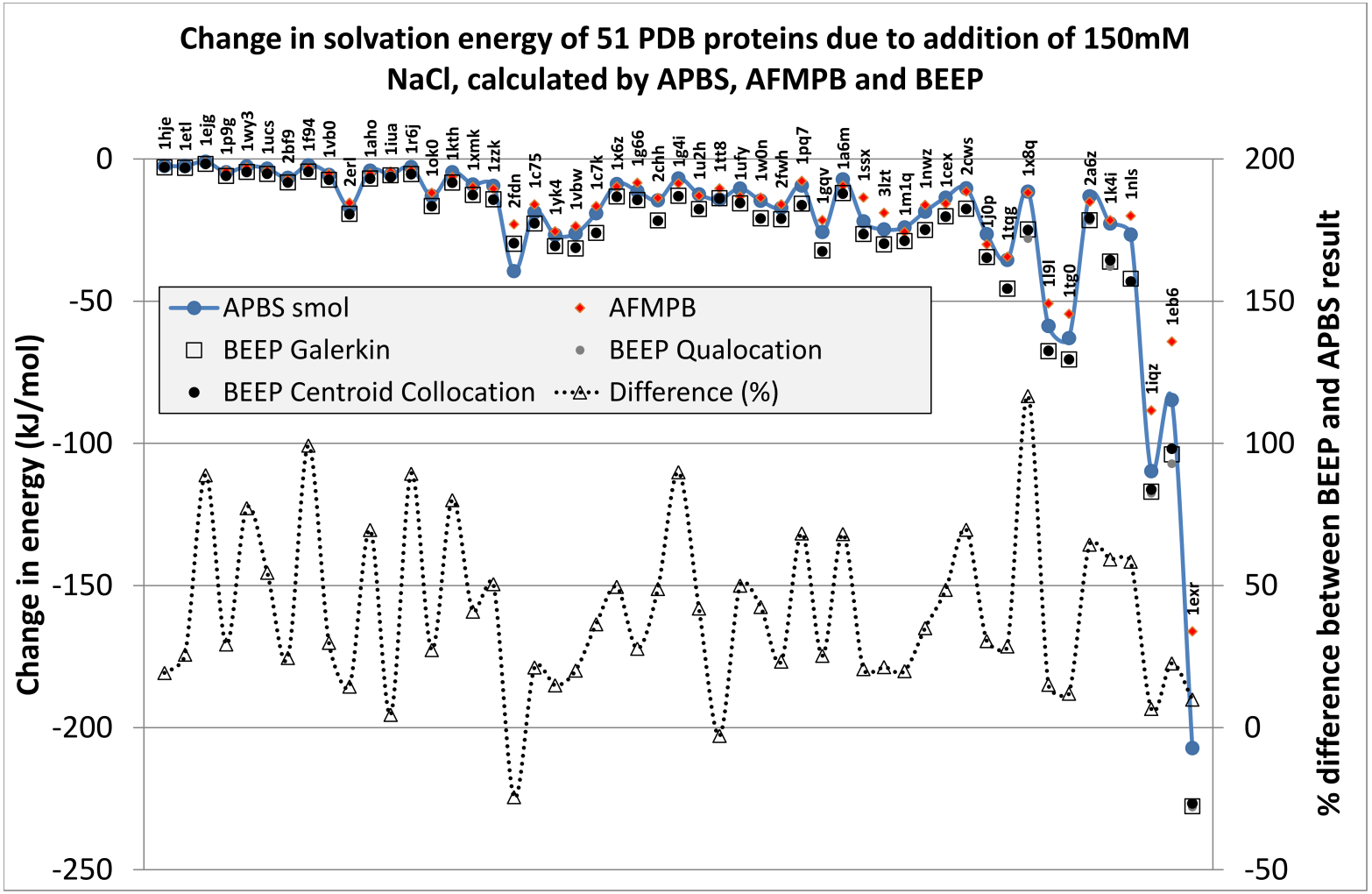


Figure 5.4: The effect of 150mM monovalent salt on electrostatic solvation energies for 51 PDB proteins (compared to zero salt), calculated using APBS, AFMPB and BEEP, with APBS ion-exclusion region defined by ionic radius of 1.5Å. Both APBS and BEEP results are converged, but give different results for what should be the same system.

with lower overall energy (i.e. the solvation energy for the ionic case becomes more negative relative to the zero salt case).

Figure 5.5 illustrates the the changes in solvation energies between zero salt and 150mM, 300mM and 450mM monovalent salt, using both BEEP and APBS, with the latter using the 0.8Å ionic radius. Under these conditions BEEP gives results that are consistent with APBS. The outliers on the graph (where the points do not lie along the 1:1 diagonal line) are where APBS gives anomalously high values for the energy (i.e. vastly in excess of the values produced when using a larger ionic radius). Presumably in these cases, the smaller ionic radius allows ions to approach highly charged parts of the protein, which a larger ionic radius in APBS would not allow, and which the solvent-excluded surface in BEEP prohibits, leading to the difference in results. From these results it seems likely that the coincident dielectric and ionic boundary within the BEM somewhat overestimates the effect of ions in the continuum solvent model.

The results of Figures 5.4 and 5.5 also show that the difference in energies obtained using each discretization scheme is almost exactly the same: the results are converged with respect to integration method, which was not the case for isolated solvation energies. It would appear that any lack in fidelity inherent in the different discretization schemes beneficially cancel out when taking the difference between two calculations, producing the same final result.

5.3 Critical Mesh Density

We have seen in previous chapters that the execution time for BEEP increases linearly with the number of node-patches. The simplest way to reduce the execution time for BEEP on a given protein or group of proteins is to reduce the number of node-patches. However, the BEM literature contains little discussion on the subject of how many mesh points are required to obtain "reliable" results. Here we use BEEP to explore the relationship between electrostatic solvation energy and mesh density. Throughout this section we use BEEP with the qualocation discretization scheme (using 4 Gauss-Legendre quadrature points per sub-triangle of each node-patch).

5.3.1 Meshing the Born Ion

We have already seen that the simplest electrostatic system is the Born Ion. Therefore we start our discussion of critical mesh density with a plot of the Born solvation energy for a single spherical ion as we decrease the number of mesh points used to describe the spherical surface. The graph of solvation energy is given in Figure 5.6a (along with a picture

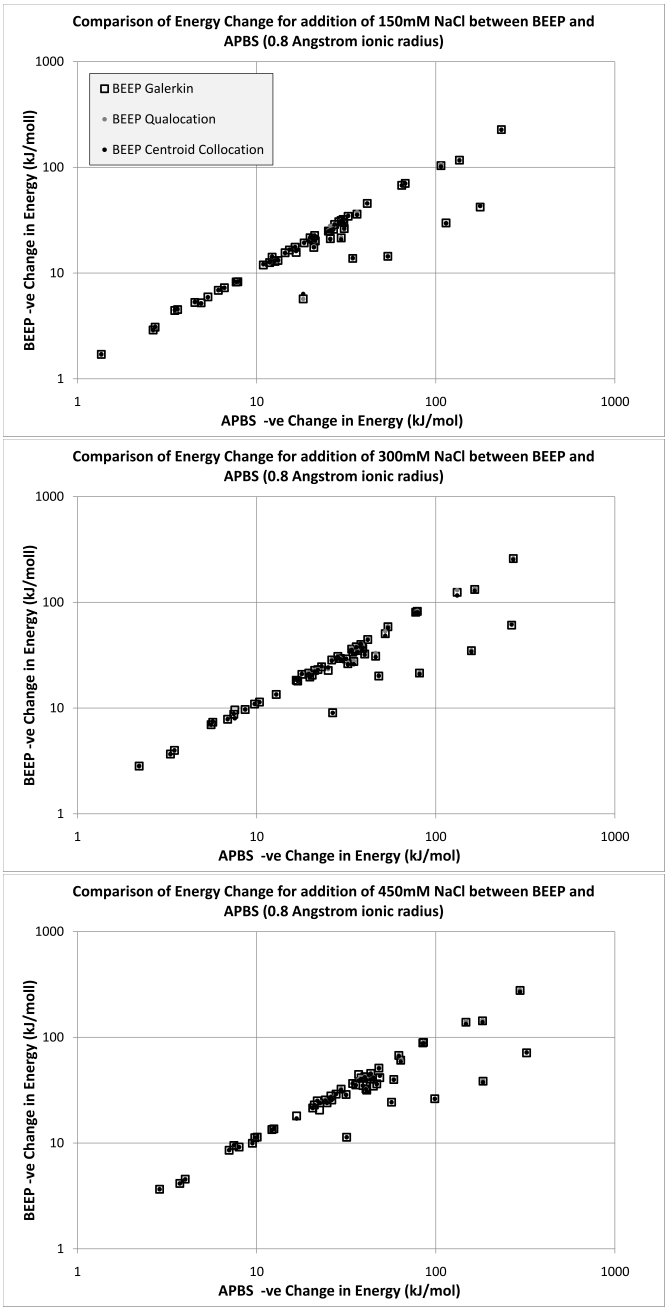


Figure 5.5: Correlation between BEEP and APBS for the change in electrostatic solvation energy (relative to zero salt) for increasing concentrations of monovalent salt for 51 PDB proteins. The APBS ion-exclusion region is defined by an ionic radius of 0.8Å. The results for all integration regimes in BEEP (Galerkin, qualocation, centroid collocation) give comparable results, suggesting that any errors or lack of fidelity inherent in each type of discretization cancel out when taking the differences between two calculations, to yield the same final result.

of the highest and lowest resolution meshes), which shows that we can obtain surprisingly good values for the solvation energy even when the mesh departs substantially from the true spherical shape. However below around 100 node-patches the solvation energy becomes sharply more negative, due to parts of the surface more closely approaching the central point charge as the mesh becomes very coarse.

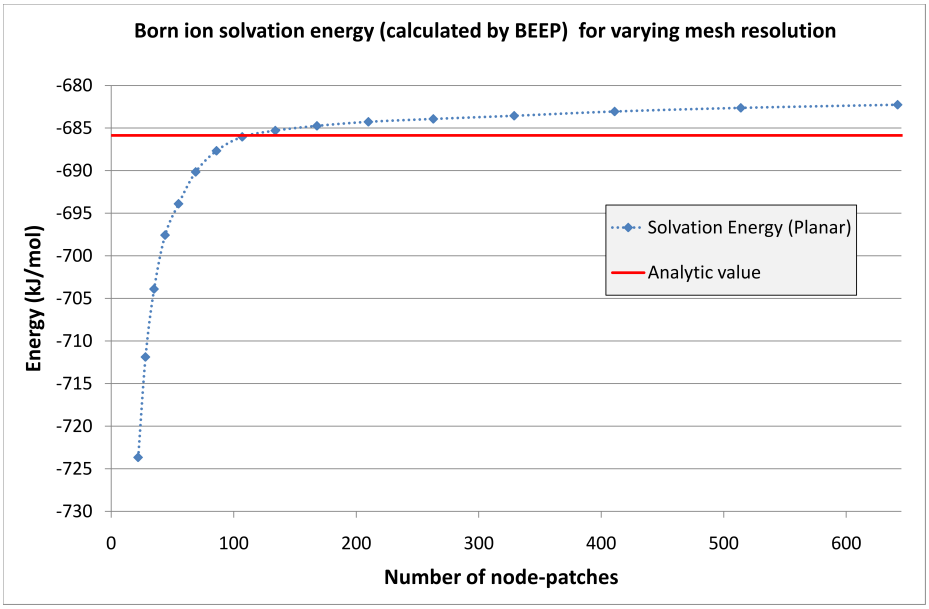
The decimation method used here aims to preserve the position of the vertices on the boundary. Alternative decimations are possible which better preserve the volume and surface area by allowing greater freedom in relocating vertices, but these will result in a surface which in places exceeds the original radius and occupies space outside of the boundary of the original mesh. Although this can improve the relative error in solvation energy (compared to that plotted in Figure 5.6a), the change in boundary position for "real" proteins may be undesirable.

5.3.2 Improved geometric accuracy using curved triangles

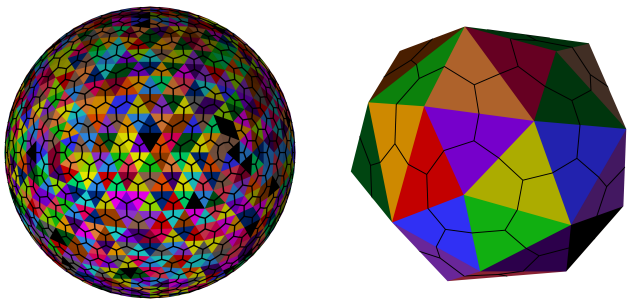
The loss of accuracy in solvation energy is due to the very poor correspondence of the low-resolution meshes to the actual geometric surface it is intended to represent. Since we expect protein surfaces to also contain regions of relatively high curvature, we thought it would be worthwhile implementing an improved triangle description to better represent the geometry of the surface for fewest numbers of elements: curved elements are the obvious answer. Indeed Bardhan *et al.* [107] have previously shown that BEM electrostatic solutions (for spheres and proteins) appeared to be improved when integrations were carried out in a coarse manner over a curved surface, as opposed to a more exact method over a planar surface .

To this end, we extended the planar node-patch method described in Chapter 3 to use a type of parametric curved triangles, called "PNG1" triangles. PNG1 is an acronym for "point-normal G1-continuous[5]", and the triangles are based on Bézier-patches. Bézier-patches represent a curved surface by interpolation of a cubic function over a set of 10 "control points", which are fixed locations in space used to define the surface. Points on a Bézier-patch are described by parametric coordinates (u, v, w) which can adopt values between 0 and 1, (with the constraint that $u + v + w = 1.0$). The parametric coordinates map to 3D points in space using Equation 5.1.

[5]Continuity of parametric surfaces can be "geometric" (G0, G1, G2...) or "parametric" (C0, C1, C2...). The numerical index refers to the highest derivative of the curve which is guaranteed continuous over the surface; parametric continuity means that the value of the derivatives are numerically equal so the surface is perfectly continuous, whilst geometric continuity guarantees only that the derivatives are proportional, i.e. that tangent vectors are in the same direction but not necessarily equal in magnitude. Creating surfaces with high-order continuity (either geometric or parametric) is non-trivial. The unit-normal vector of a G1-continuous surface is continuous.



(a) Electrostatic solvation energy compared to analytic value as function of mesh density: low values on the x-axis are few vertices per unit area, and are the lowest resolution meshes. The solution can be seen to converge well for higher mesh resolutions, though it does not actually match the analytic value.



(b) The high and low resolution meshes (642 vertices vs. 22 vertices) showing the original planar triangles, and the corresponding node-patches (black outlines): the low resolution mesh does not appear very spherical, so it is not surprising that the corresponding value for solvation energy is very inaccurate compared to the analytic value.

Figure 5.6: The Born Ion for various mesh resolutions

$$
\begin{aligned}
p(u,v,w) &= b_{300} \times w^3 + b_{030} \times u^3 + b_{003} \times v^3 \\
&+ b_{210} \times 3w^2u + b_{120} \times 3wu^2 + b_{201} \times 3w^2v \\
&+ b_{021} \times 3u^2v + b_{102} \times 3wv^2 + b_{012}3 \times uv^2 \\
&+ b_{111} \times 6wuv \qquad (5.1)
\end{aligned}
$$

where b_{abc} represent the control points. The gradient at any point with respect to each parametric coordinate can be used to define a normal vector at that point. The control points b_{300}, b_{030}, b_{003} generally coincide with the vertices of the planar triangle on which the curved surface is based, whilst the placement of the remaining points controls the variation in curvature over the surface. In PNG1 triangles some of the control points are in fact *not* fixed but are themselves dependent on the values of u and v and blend together information from neighbouring triangles to ensure that the resultant surface is G1-continuous.

PNG1 triangles are described fully by Fünfzig *et al.* [149]; the full details of the method are not particularly relevant here, so we merely state that we implemented PNG1 triangles in BEEP, in an attempt to create a surface mesh with better continuity of normal vector than planar triangles offer [6].

The PNG1 triangles can be easily built from the original planar triangle mesh, and any point on the new curved surface can be found by parametric interpolation into the corresponding PNG1 triangle. Curved node-patches can then be built in exactly the same way as with planar triangles but using PNG1 triangles instead: quadrature points on node-patches which were previously defined for a collection of planar sub-triangles can be mapped directly onto the curved triangles, with an appropriate change in quadrature weight to reflect the affine transformation in changing from planar triangle to curved PNG1 triangle [7].

PNG1 triangles introduce a small overhead into BEEP in that they must be created from the planar triangles at the start of the program, and calculating the quadrature points from parametric coordinates is more costly in terms of floating point operations than the same procedure on planar triangles, because each parametric point is found by evaluating a set of cubic equations involving multiple control points rather than simple linear interpolation. However, in the larger context of the entire BEM/FMM algorithm this increase in computation is negligible.

[6] Our original motivation was to improve estimation of electric field vector from our surface solutions, but as we will see shortly that did not work out as intended.

[7] multiplication by the determinant of the Jacobian matrix, which corresponds to the combination of partial derivatives of position w.r.t the coordinates u,v given by: $\sqrt{EG-F^2}|_{p(u,v,w)}$ where $E = \left(\frac{\partial x}{\partial u}\right)^2 + \left(\frac{\partial y}{\partial u}\right)^2 + \left(\frac{\partial z}{\partial u}\right)^2$,$F = \left(\frac{\partial x}{\partial u}\frac{\partial x}{\partial v}\right) + \left(\frac{\partial y}{\partial u}\frac{\partial y}{\partial v}\right) + \left(\frac{\partial z}{\partial u}\frac{\partial z}{\partial v}\right)$, $G = \left(\frac{\partial x}{\partial v}\right)^2 + \left(\frac{\partial y}{\partial v}\right)^2 + \left(\frac{\partial z}{\partial v}\right)^2$. (See also Zhao *et al.* [84]).

The advantages of PNG1 triangles are expected to be two-fold:

1. Curved triangles allow us to represent a surface with fewer node-patches than would be possible with planar triangles. We would expect that this would lead to reduced "surface error" when lowering the mesh density of a given protein surface.

2. PNG1 triangles give a continuous surface-normal vector instead of piecewise constant normal. We suggest that this should improve the correspondence between the discretized system of equations and the continuous system of boundary integral equations, meaning that BEM solutions for a given level of discretization should have slightly higher fidelity with respect to the "true" continuous solutions. In particular the BEM kernel functions B_{pt} and C_{pt} (which are derivatives of Green's functions with respect to surface normal vectors) evaluate to zero when taken over a single planar triangle (as occurs when carrying out the near-singular "self-patch" integrations). Consequently, under a planar geometry, the self-patch integrals contain a numerical component which evaluates to zero but which for the "real" curved surface should be non-zero. Through using curved triangles, these kernel functions will be evaluated with higher fidelity. The numbers we are referring to appear as part of the dominant diagonal coefficients in the BEM matrix. A small change in these values could lead to a proportionate change in the final solution, which would not be the case for the less critical off-diagonal matrix components.

Figure 5.7 shows the improvement in solvation energy produced by using PNG1 triangles to represent a spherical surface. Not only is the limiting behaviour now equal to the analytical results, even for the very coarse 22 vertex "sphere" the solvation energy is within $\sim 1\%$ of the analytic value. Intuitively, we would also expect curved elements to have a similarly beneficial effect on the representation of highly curved protein surfaces, with consequential improvement in representation of the surface within the BEM (and therefore, hopefully, higher fidelity in our solutions).

5.3.3 Critical meshing of acetylcholinesterase and fasciculin

5.3.3.1 Acetylcholinesterase and fasciculin

To demonstrate the effect of mesh density on solvation energy we choose the proteins acetylcholinesterase (ACHE) and fasciculin (FAS). Acetylcholinesterase is an enzyme which is secreted between synapses and ordinarily breaks down the neurotransmitter acetylcholine. Certain snake venoms contain a small neurotoxic protein called fasciculin which binds very strongly to, and inhibits the enzymatic activity of, acetylcholinesterase. Both fasciculin and

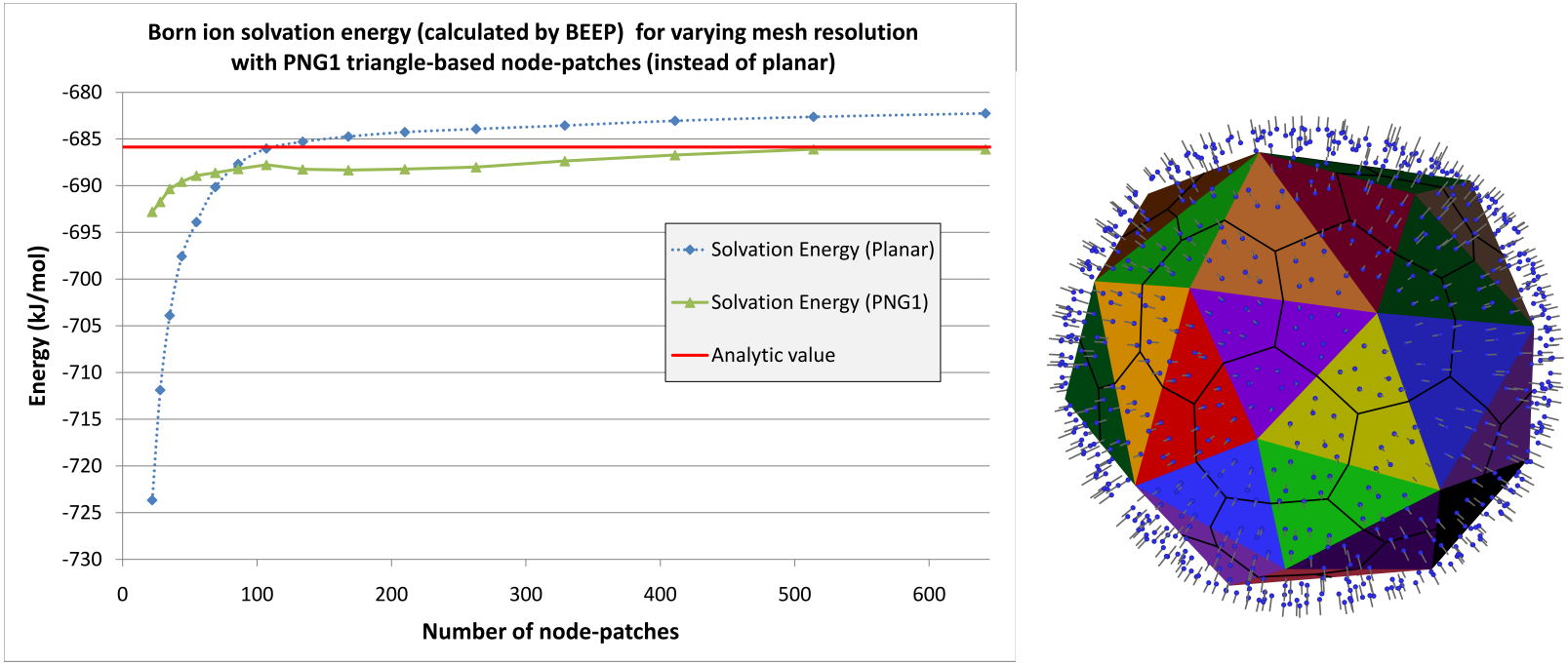


Figure 5.7: Improved solvation energy for Born Ion using PNG1 Triangles, with an illustration of the PNG1 triangle surface integration points (blue does, with short lines representing the normal vector at the point). The underlying 22 vertex planar triangular mesh is the same as that in Figure 5.6b. The solvation energy converges on the correct value for high resolution meshes (suggesting we have gained accuracy somewhere in the numerical integration process) and remains within 7 kJ/mol of the analytic value even at very low mesh densities.

acetylcholinesterase are relatively highly charged proteins and their interaction is strongly driven by electrostatic complementarity, as discussed by Elcock *et al.* [54]. Consequently, the acetylcholinesterase/fasciculin system is well studied in the protein electrostatics literature (particularly by the McCammon group [150–153]), and we will be using it as a convenient test system for bi-molecular electrostatic interactions.

Our structures for (mouse) acetylcholinesterase and fasciculin are taken from the PDB structure 1MAH, in which the two proteins are bound in a complex. HETATOM entries were removed, and the two structures separated into two individual PDB-format files. Charges and radii were then assigned using PDB2PQR and the PARSE forcefield (as for the other 51 test proteins earlier in this chapter). Dielectric constants were set to 2.0 for proteins and 80.0 for the solvent, and salt concentration was set to zero. Surface meshes were built using MSMS (solvent probe radius of 1.5Å) followed by Meshlab rebuilding and refinement, as outlined in Appendix B.

5.3.3.2 Solvation energies

Figure 5.8 plots the total solvation energy as a function of mesh detail for the acetylcholinesterase and fasciculin, using both planar and PNG1 triangles. These results are less impressive than the improvement obtained for the Born Ion. There is no consistent benefit to using the PNG1 triangles. The minimum mesh density to obtain agreement within 5% of the converged value of solvation energy are 0.6 vertices/$\mathring{A}^2$ (PNG1 triangles) vs. 0.75 vertices/$\mathring{A}^2$ (planar) for acetylcholinesterase. For fasciculin the minimum mesh density for the same level of "accuracy" is 0.65 vertices/$\mathring{A}^2$ for PNG1 triangles, and 0.45 vertices/$\mathring{A}^2$for planar,

At first this lack of improvement from the curved surface representation seems strange. However, if we consider the various sources of error in the calculation of solvation energy (relative to some hypothetical "perfect" solution) we can demonstrate what is happening in this case.

5.3.3.3 Fidelity vs. the change in dielectric boundary

The effect of reducing mesh density is two-fold:

1. Reducing the number of mesh points lowers maximum fidelity of the solution, by which we mean that as mesh resolution decreases the solution looks increasingly "blocky" and its correspondence to the "true" continuous solution (i..e that which could be found by an arbitrarily finely divided mesh) is reduced. We refer to this effect as the "fidelity error".

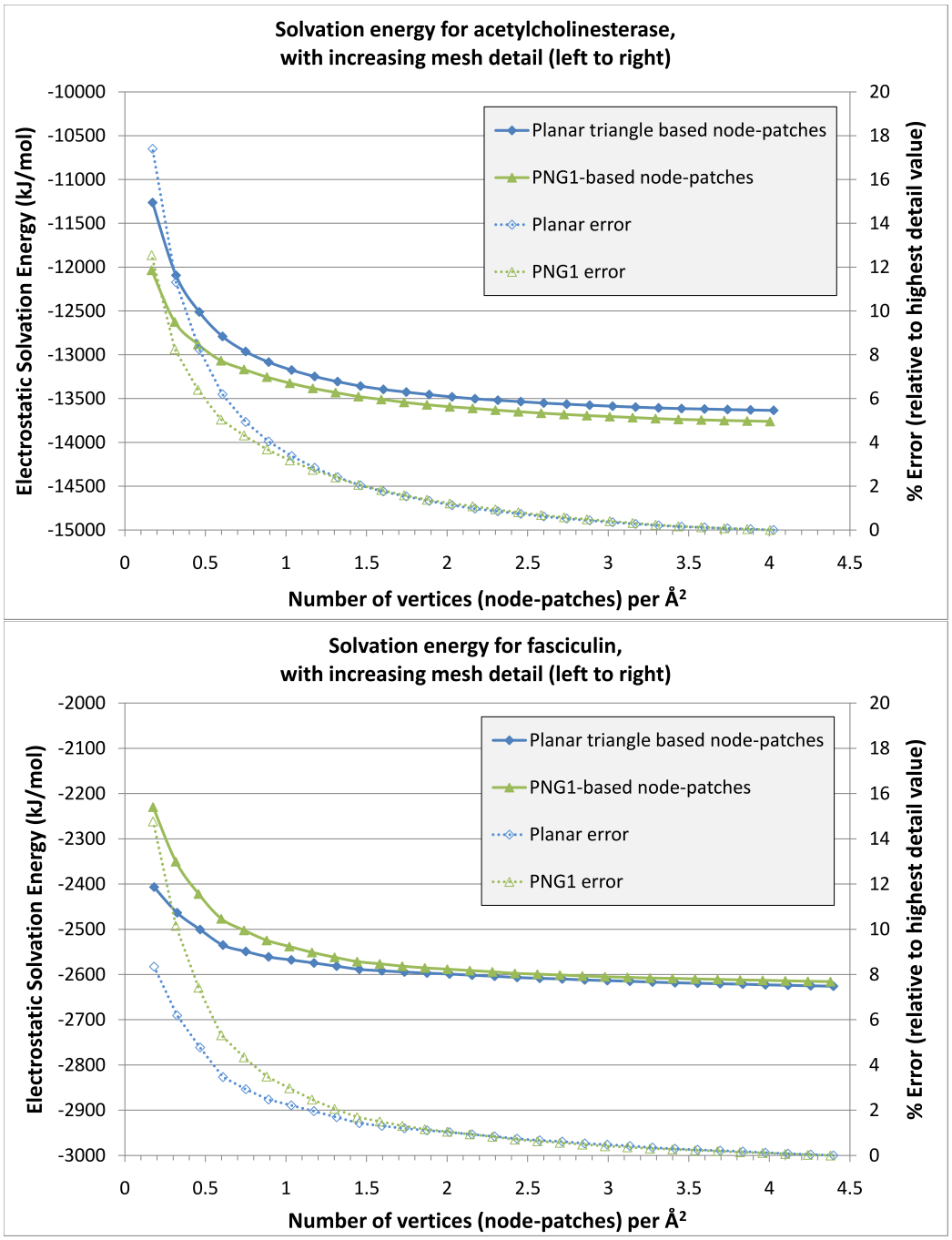


Figure 5.8: The solvation energy of acetylcholinesterase and fasciculin, for varying mesh densities

2. Reducing the number of mesh points generally results in a change in the definition of the surface, such that the dielectric/ionic boundaries lie in a different place and surface elements may cut through space closer to atomic charges. This results in a change in the model being solved, in addition to the reduction in points at which the numerical solution is found. We will refer to the change in solvation energy due to the alteration of dielectric boundary definition as "surface error".

The effect of fidelity error can be seen visually in Figure 5.9, which gives the surface solution for fasciculin for three mesh resolutions: the degradation in solution detail for low mesh resolution can be seen by eye. The error in solvation energy for the lowest resolution mesh in Figure 5.9 is only 8%, despite the apparently quite coarse representation of the surface solution.

In general when reducing the mesh density we will observe a change in solvation energy which is due to the combination of these two errors. In our trivial example of the Born Ion, the surface solutions are (analytically, on a perfect sphere) constant values over the surface, which is very simple to match in terms of fidelity using constant-value elements. The breakdown in the solvation energy in that example was purely geometric surface error, which was remedied by using PNG1 triangles. In our example for fasciculin, perfect fidelity of the solution is not so simple to achieve: as we can see from the visualisations of the surface potential and normal field component, the surface distributions are complicated and in certain places the solution varies rapidly over the surface.

We have attempted to quantify the extent to which the change in solvation energy plotted in Figure 5.8 is due to these "fidelity error" and "surface error" components, by recalculating solvation energies for the lower resolution meshes, with additional vertices derived from a higher resolution mesh. By projecting vertices from a high-resolution mesh onto the "poor" low resolution surfaces (for example, using the vertices from the 2 vertices/$\mathring{A}^2$ mesh, which Figure 5.8 suggests should contain less than 1% total error compared to the "converged" value). we can calculate the solvation energy of the low resolution surfaces without incurring any fidelity error[8].

Once we have a solvation energy for a mesh with "zero" fidelity error, we calculate the difference between this and the point on Figure 5.8 corresponding to the same surface; the difference will be an estimate for the fidelity error. We can also estimate the surface error,

[8]This assumes that the change in surface has not resulted in a set of surface potentials and normal field components which cannot be adequately represented by the "high fidelity" number of node patch solution points. This assumption may well break down at very low mesh resolutions when surface elements could be so close to atomic charges that the surface solutions have a functional form which varies across the surface more rapidly than 2 vertices per $\mathring{A}^2$ can adequately capture.

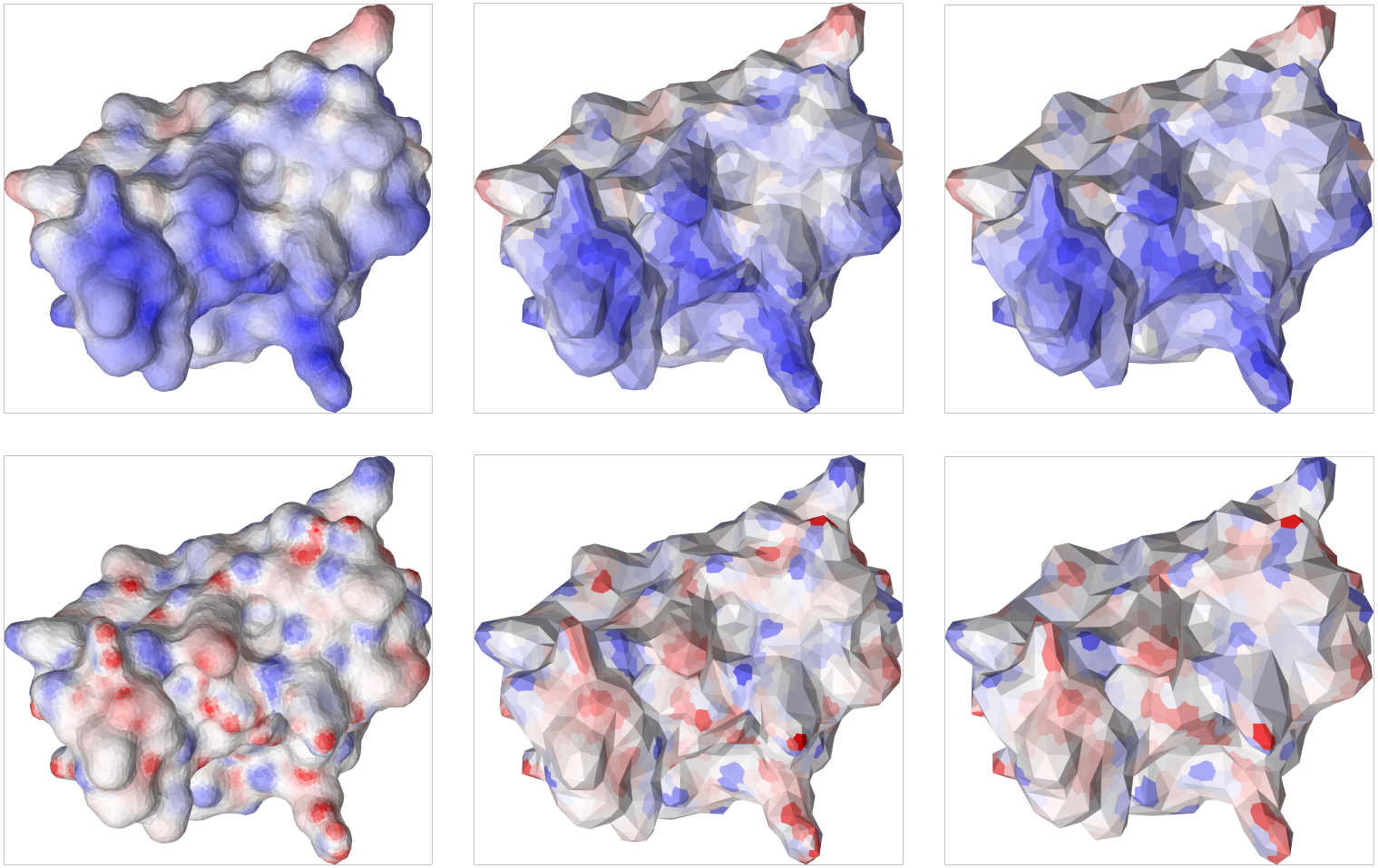


Figure 5.9: Visualisations of the surface solutions for fasciculin, as mesh resolution is decreased: colours reflect the magnitude of the potential (top row) and normal electric field (bottom row). The latter is more important in computing the electrostatic solvation energy. The mesh densities are 4.4 vertices/$\mathring{A}^2$ (left), 0.32 vertices/$\mathring{A}^2$ (middle) and 0.18 vertices/$\mathring{A}^2$ (right). The relative error in solvation energy of the middle and right-hand meshes, compared to the high-resolution version on the left, are 5.8% and 8.0%.

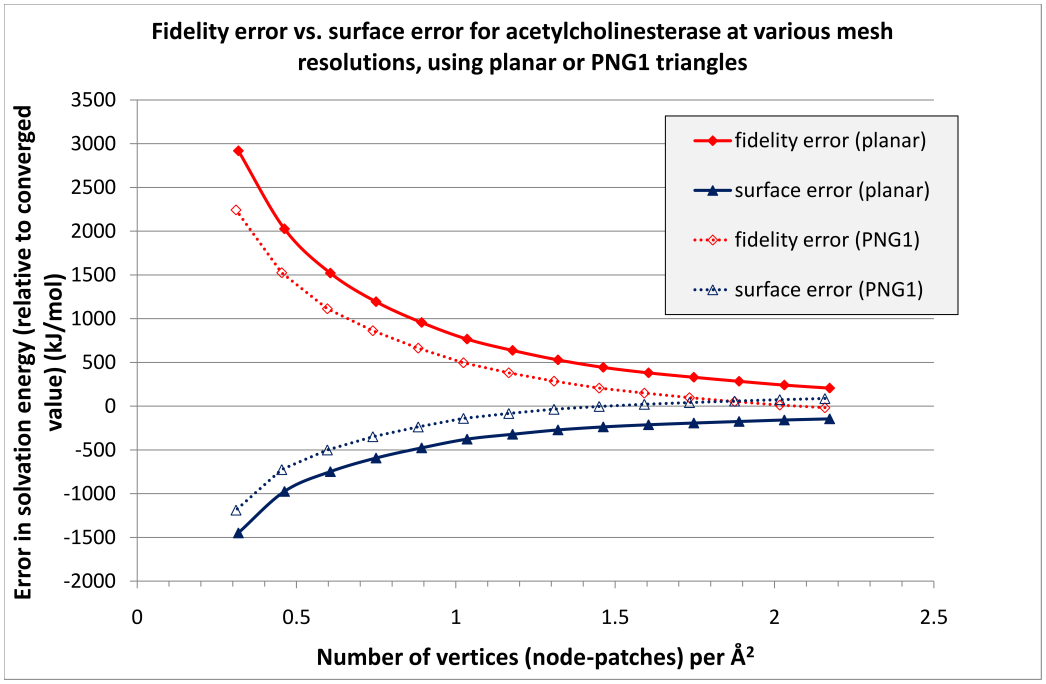


Figure 5.10: "Fidelity error" vs. "surface error" for acetylcholinesterase and fasciculin

by subtracting the energy for the high-fidelity/poor-surface combination from the result for the high-fidelity/good-surface (i.e. 2 vertices/$\mathring{A}^2$ the reference mesh).

Estimates for the fidelity error and surface error for acetylcholinesterase, for both planar and PNG1 triangle representations, are plotted in Figure 5.10. The surface error induces a small increase in the magnitude of the solvation energy (i.e. more negative), since the reduction in surface points tends to contract the surface, lowering the distance from charges to the surface. The fidelity error is in the opposite direction, and is a greater effect, by approximately a factor of two.

Interestingly, this plot suggests that the surface error counteracts some of the fidelity error in this case, resulting in a rather slow loss of accuracy with decreasing numbers of planar triangles. Our PNG1 triangles improve the error in both parts by approximately the same absolute amount leading to little net gain. Although we have made an improvement in the geometric representation of the surface, because of this error cancellation we don't obtain a marked improvement in solvation energy.

The surface error is always likely to make solvation energies more negative due to the contraction of the surface. We have no obvious explanation for why the fidelity error should act in the opposite direction. The solvation energy is dominated by the results for surface field component (h) because it is multiplied by the ratio of dielectric constants, so it seems reasonable to suppose that the fidelity error in the case of acetylcholinesterase (and fasciculin, which has a similar curve to Figure 5.10) corresponds to a decrease in the average value of

h over the surface. Since the origin of the fidelity error is the coarsening of the electrostatic surface solutions (analogous to quantization error in signal processing fields), presently we can think of no reason why this should lead to a change in the average magnitude of electric field values over the surface.

5.4 Calculating Intermolecular Electrostatic Forces

The above section demonstrated that BEEP can solve the electrostatic solvation energy for a protein to good accuracy, with a possible trade-off between mesh density and "accuracy" by which we mean similarity to a highly-detailed surface solution. However if we wish to use BEEP as the basis for a biomolecular simulator (using, for example, Brownian Dynamics), it is the intermolecular forces which could be of more use, rather than solvation energy, so we now turn our attention to the electrostatic force components introduced in Chapter 3. The test system is again acetylcholinesterase and fasciculin with the same experimental conditions as above (protein dielectrics=2.0; solvent dielectric=80.0; zero salt, i.e. $\kappa = 0$, unless otherwise stated).

5.4.1 Electrostatic force on an isolated protein

Table 5.3 shows magnitude of the net electrostatic force acting on acetylcholinesterase and fasciculin (separately, in isolation) for some of the well-converged meshes used above (and also some results for coarse meshes, for comparison). The calculation is analogous to that given in Chapter 3 for spherical test cases: the force comprises a qE force of the reaction field acting on the source charges, plus a dielectric boundary force and an ionic pressure. The qE force can be calculated either by integration of the Maxwell Stress Tensor (MST) on the *inside* of the dielectric boundary or by the "patch-charge" method. The other components (dielectric boundary force (dbf) and ionic pressure) can only be found by numerical integration of the surface solutions, via interpolation of the electric field vector from BEM surface solutions. Therefore the net force (qE + dbf + ionic) has two possible values here, depending on whether one chooses the patch-charge or MST to calculate qE force.[9].

[9] As we explained in Chapter 3, the dielectric boundary force can be expressed in two ways (which are mathematically equivalent): firstly, from the derivation of Gilson *et al.* [136], we can write the dielectric boundary force as a volume integration of energy density of the dielectric boundary: $\int -\frac{1}{2} E.E \nabla \varepsilon dV$. This integral involves the gradient of the dielectric over the surface layer: since the surface layer is infinitely thin it has infinite gradient this is problematic. Alternatively (and rather more pragmatically) the dbf can be viewed as the difference in stress as represented by the Maxwell Stress Tensor evaluated on the outside vs. the inside of the dielectric boundary. This leads to a more useful expression for dielectric boundary force: $-\frac{1}{2}(\varepsilon_{ext} - \varepsilon_{int}) \times (E_{int} \cdot E_{ext})\hat{n}$. We have chosen to separate the qE forces and dbf forces into separate parts so that we can assess the relative contribution of each to the total force: in practice the total force on the entire molecule can be found by a single evaluation of the Maxwell Stress Tensor on the *outside* of the

In either case, we must make use of the electric field vector, E, constructed from the BEM surface solutions by a singular-value decomposition method.

Whilst the test-cases of Chapter 3 gave tolerably accurate results (i.e. the net force reduced to nearly zero for a reasonably small number of mesh vertices), here we see that the force calculations break down for real protein surfaces. The net force on an isolated protein should be zero, but it is distinctly non-zero for all cases here, using either the patch-charge and MST-based qE components. It seems likely from the results of Chapter 3 that the qE force calculated by the patch-charge method is more "accurate", but the commonality of sources of error in the MST and dielectric boundary force has the effect that the net force calculated using qE forces derived from the MST are closer to zero than the net forces based on the patch-charge force.

The origin of the error in the total force is our inability to calculate the different force components with sufficient accuracy, which are individually of moderately large magnitude, but should theoretically cancel out[10]. This lack of accuracy in force components stems from our inability to accurately estimate E. Recall Figure 3.11 in Section 3.3.3 of Chapter 3 which illustrated the difficulty we faced in calculating accurate force components, even for a "simple" spherical surface with off-centre charge: the analogous problem for real protein surfaces has proven just as severe, in which many charges are close to the non-spherical boundary. Our optimism regarding local curvature and the scaling of the error in Figure 3.11 of Chapter 3 has proven to be unfounded.

PNG1 triangles do not necessarily give lower net force magnitudes than planar triangles, even though we have some reason to believe that the quality of the overall solution (i.e. overall fidelity relative to very high resolution solution) should have been improved using that method (Figure 5.10). Clearly the extraction of the electric field vector over the curved triangular elements (chosen primarily for their property of continuous surface normal vector) does not does not have any benefit on the final summation of forces: in fact the amount of error in force (i.e. deviation from zero) generally increases using the PNG1 triangles, suggesting that the electric field vectors contain even more numerical error than when using

dielectric boundary, and then adding the additional ionic force. This is the more usual method in the BEM literature.

[10]We are confident that this is *not* merely a result of numerical error arising from the limits of double precision floating-point arithmetic for two reasons: firstly we use Kahan compensated addition to ensure that we do not suffer from floating-point accumulation errors. Secondly we can directly calculate the combination of $qE_{MST} + dbf$ by use of the Maxwell Stress Tensor on the *outside* of the protein. Since the qE terms and dbf terms are contained within a single expression on each node-patch, the result is well-behaved sum of small numbers over the whole surface, and the need to find the difference between two moderately large numbers by floating point subtraction is avoided. The result is identical in both cases: it is not our decision to split forces into qE and dbf components which causes the problem.

				vertices per Å^2	Force Components (kJ/mol.Angstrom)																Total Force (patch-charge + dbf) (kJ/mol.Angstrom)				Total Force (MST + dbf + ionic) (kJ/mol.Angstrom)			
					qE force								Dielectric boundary force (dbf)				Ionic force											
					patch-charge method				Maxwell Stress Tensor																			
					x	y	z	**mag.**	x	y	z	**mag.**	x	y	z	**mag.**	x	y	z	**mag.**	x	y	z	**mag.**	x	y	z	**mag.**
0mM Salt (k=0)	**FAS**	Planar	very high detail	4.4	-182	53.62	-122	**225.5**	-173	49.44	-110	**210.9**	172.7	-49.6	110.5	**211**					-9.18	3.981	-11.5	**15.27**	-0.16	-0.2	0.37	**0.452**
			recommended	1.3	-178	54.49	-119	**221.2**	-162	33.31	-94.1	**190**	161.9	-33.6	94.69	**190.5**					-16.5	20.87	-24.2	**35.95**	0.2	-0.3	0.557	**0.665**
		PNG1	very high detail	4.4	-179	51.17	-110	**216.2**	-172	48.32	-105	**207**	171.1	-48.6	106.6	**207.3**					-7.93	2.565	-3.26	**8.949**	-0.91	-0.28	2.062	**2.273**
			recommended	1.3	-179	48.93	-106	**213.9**	-164	33.48	-90.1	**189.7**	163.6	-34.2	91.72	**190.6**					-15.4	14.75	-14.7	**25.91**	-0	-0.69	1.627	**1.768**
	ACHE	Planar	very high detail	4.0	-915	-44.8	321.6	**970.7**	-870	-35.6	306.2	**922.8**	864.4	71.29	-338	**931**					-50.4	26.45	-16.7	**59.29**	-5.36	35.73	-32.1	**48.32**
			recommended	1.5	-892	-15.3	301	**941.2**	-806	-12.2	286.6	**855.8**	818.6	8.116	-285	**866.9**					-73	-7.23	15.79	**75.07**	12.28	-4.12	1.408	**13.03**
		PNG1	very high detail	4.0	-873	-52.1	294.2	**923**	-843	-46.3	295.5	**894.3**	887	29.27	-296	**935.5**					13.65	-22.8	-1.63	**26.64**	44.16	-17	-0.27	**47.34**
			recommended	1.5	-873	56.61	297.1	**923.6**	-785	-19.9	278.6	**833.1**	818.6	6.653	-284	**866.4**					-54	63.26	13.36	**84.26**	33.74	-13.2	-5.08	**36.6**
150mM Salt (k=0.127)	**FAS**	Planar	very high detail	4.4	-182	53.65	-122	**225.5**	-173	49.41	-110	**210.8**	172.9	-49.8	110.6	**211.2**	-0.08	-0.07	0.056	**0.125**	-8.8	3.814	-11.6	**15.09**	0.218	-0.43	0.258	**0.545**
			recommended	1.3	-178	54.41	-119	**221**	-161	33.2	-94.2	**189.8**	161.9	-33.7	94.71	**190.6**	-0.08	-0.06	0.055	**0.119**	-16.3	20.66	-24.3	**35.78**	0.452	-0.55	0.519	**0.881**
		PNG1	very high detail	4.4	-179	51.09	-110	**216**	-171	48.08	-105	**206.7**	171.9	-49	106.5	**208**	-0.08	-0.07	0.058	**0.118**	-6.82	2.049	-3.69	**8.019**	0.287	-0.96	1.66	**1.94**
			recommended	1.3	-179	48.7	-107	**213.6**	-163	33.06	-90.2	**189.3**	164	-34.5	91.6	**191**	-0.07	-0.06	0.059	**0.109**	-14.6	14.19	-15	**25.32**	0.925	-1.45	1.406	**2.224**
	ACHE	Planar	very high detail	4.0	-912	-36.2	327.5	**969.4**	-867	-27.3	313.5	**922.3**	864.6	33.11	-320	**922.5**	2.467	-0.5	-2.3	**3.407**	-44.6	-3.58	5.173	**45.02**	0.121	5.335	-8.79	**10.28**
			recommended	1.5	-889	-6.4	307.6	**940.5**	-804	-2.08	293.4	**856.1**	806.1	0.617	-293	**857.6**	2.445	-0.54	-2.24	**3.363**	-80.2	-6.33	12.53	**81.4**	4.338	-2.01	-1.65	**5.055**
		PNG1	very high detail	4.0	-873	-46.2	299.9	**924.3**	-844	-38.9	302.5	**897.7**	856.7	33.07	-307	**910.7**	2.225	-0.35	-2.16	**3.119**	-14.1	-13.5	-9.51	**21.7**	14.63	-6.14	-6.92	**17.3**
			recommended	1.5	-874	62.66	302.6	**926.6**	-787	-12.6	285.4	**837.4**	797.8	7.638	-290	**849**	2.149	-0.34	-2.06	**2.998**	-73.7	69.96	10.32	**102.1**	12.8	-5.27	-6.89	**15.46**

Table 5.3: Net forces on isolated molecules of acetylcholinesterase and fasciculin, for zero salt conditions and at 150mM monovalent salt concentration. Here we are using the SVD method to extract the electric field vector from surface potential/field as described in Section 3.2.3.3 of Chapter 3.

planar triangles. From this we conclude that our failure to extract electric fields is not (only) due to insufficiently smooth geometric representation of the surface.

Increasing the detail of the surfaces from "recommended" (which we chose as the surfaces which gave a solvation energy within 2% of the "converged" value in Figure 5.8) to the "very high detail" gives mixed results: the total force using the patch-charge qE force tends to improve significantly, but the total force using the MST qE force actually gets worse. The improvement in patch-charge based total forces is most pronounced for the PNG1 case and can be seen as resulting from a change in the dielectric boundary force, since the patch-charge qE force appears very well converged. The decrease in accuracy of the MST-based force arises from increased numerical error in that calculation: increased numbers of surface points results in increased numbers of inaccurate force components to add together (based on a larger number of inaccurate estimates for electric field). On the other hand increased mesh points does improve dielectric boundary forces, even though it is also based on the electric field vector E. We suggest that this is because the dielectric boundary force is more dependent on the normal components of the electric field than the planar components, unlike the MST which is a complicated function of the field vector components. This would explain why the improvement with mesh detail is not very large, and implies that the dielectric boundary force is unlikely to ever converge on the "correct" value (cancelling the patch-charge qE force) even with very detailed meshes, as the planar components of electric field will always hamper the accuracy.

To summarise, the total force is limited by the accuracy with which we can calculate the dielectric boundary force, which is a function of the total electric field vector at the surface, comprising the normal component found by the BEM, and planar components which must be estimated by interpolation. As we have stated previously, we are attempting to extract a continuously varying quantity (the planar components of electric field at various points over each surface triangle) from a set of values for potential at vertices which were generated on the assumption that they do *not* vary over the triangle (i.e. the underlying discretization of the BEM equations assumes that the quantity we are trying to estimate is zero).

5.4.2 Electrostatic force between two proteins

We have also tried to calculate the intermolecular force between acetylcholinesterase and its inhibitor fasciculin, by placing the proteins adjacent in space and running BEEP. We used the two details of surface mesh from the previous section with node-patches based on both planar and PNG1 triangles. The protein and ligand in each case were taken from PDB x-ray crystal structures of the complex (1MAH): the protein and ligand were placed such that the geometric centres of the protein/ligand were separated along the centre-centre axis

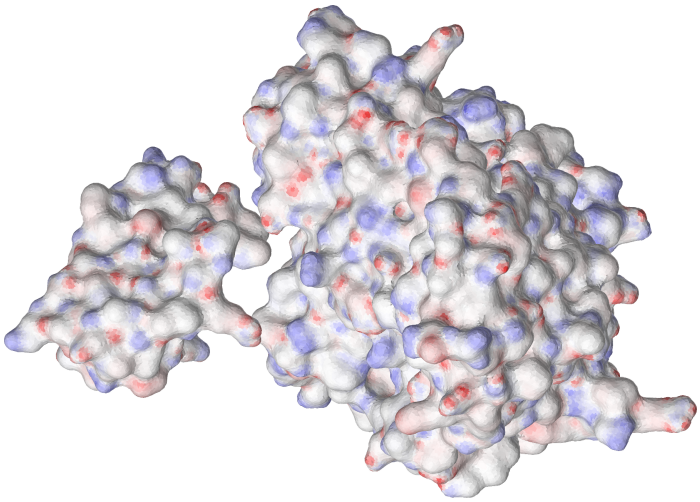


Figure 5.11: The layout in space of acetylcholinesterase (background) and fasciculin (foreground) (coloured according to surface potential)

from the original crystal structure, such that there was 6Å of solvent between the closest atoms of the two structures (assuming a water molecule radius of 1.5Å, this is equivalent to being able to fit up to 2 water molecules between the closest points of approach). The centre-centre distance between the molecules for 6Å solvent separation was 39.5Å, which is a spatial translation of 10.2Å relative to the bound complex. This is illustrated for each protein/ligand pair in Figure 5.11.

The solvation energies and forces calculated on each pair of proteins is given in Table 5.4.

Predictably enough, the forces on each protein are dominated by the enormous residual force which we observed in the previous section. In Chapter 3 we saw that we could recover the accuracy of the force between a pair of spherical meshes by subtracting the residual force calculated for the isolated cases. Carrying out the same procedure for these protein/ligand pairs does not yield sensible results: the net force on the whole system should be zero, and the forces on the protein/ligand should be equal and opposite, which is clearly not the case here.

Interestingly it no longer seems to matter very much which method is used to calculate qE force: the results for the net "corrected" force is between 1.8 and 2.9 kJ/mol.Å for fasciculin and between 4 and 20 kJ/mol.Å for acetylcholinesterase (depending on surface detail and type of element used). For acetylcholinesterase the net force is dominated by the dielectric boundary force, whereas for fasciculin the qE force has the greater contribution to total force. It seems plausible that the change in dielectric boundary force is extremely noisy in the case of "very high detail" acetylcholinesterase, due to the large number of summations of inaccurate force estimates. The better estimate for the dielectric boundary force may in

				vertices per Å^2	Force Components (kJ/mol.Angstrom)												Total Force (patch-charge + dbf) (kJ/mol.Angstrom)				Total Force (MST + dbf) (kJ/mol.Angstrom)			
					qE force								Dielectric boundary force (dbf)											
					patch-charge method				Maxwell Stress Tensor															
					x	y	z	mag.	x	y	z	mag.	x	y	z	mag.	x	y	z	mag.	x	y	z	mag.
Isolated	Planar	Very high detail	FAS	4.4	-182	53.62	-122	**225.5**	-173	49.44	-110	**210.9**	172.7	-49.6	110.5	**211**	-9.18	3.981	-11.5	**15.27**	-0.16	-0.2	0.37	**0.452**
			ACHE	4.0	-915	-44.8	321.6	**970.7**	-870	-35.6	306.2	**922.8**	864.4	71.29	-338	**931**	-50.4	26.45	-16.7	**59.29**	-5.36	35.73	-32.1	**48.32**
		Recommended	FAS	1.3	-178	54.49	-119	**221.2**	-162	33.31	-94.1	**190**	161.9	-33.6	94.69	**190.5**	-16.5	20.87	-24.2	**35.95**	0.2	-0.3	0.557	**0.665**
			ACHE	1.5	-892	-15.3	301	**941.2**	-806	-12.2	286.6	**855.8**	818.6	8.116	-285	**866.9**	-73	-7.23	15.79	**75.07**	12.28	-4.12	1.408	**13.03**
	PNG1	Very high detail	FAS	4.4	-179	51.17	-110	**216.2**	-172	48.32	-105	**207**	171.1	-48.6	106.6	**207.3**	-7.93	2.565	-3.26	**8.949**	-0.91	-0.28	2.062	**2.273**
			ACHE	4.0	-873	-52.1	294.2	**923**	-843	-46.3	295.5	**894.3**	887	29.27	-296	**935.5**	13.65	-22.8	-1.63	**26.64**	44.16	-17	-0.27	**47.34**
		Recommended	FAS	1.3	-179	48.93	-106	**213.9**	-164	33.48	-90.1	**189.7**	163.6	-34.2	91.72	**190.6**	-15.4	14.75	-14.7	**25.91**	-0	-0.69	1.627	**1.768**
			ACHE	1.5	-873	56.61	297.1	**923.6**	-785	-19.9	278.6	**833.1**	818.6	6.653	-284	**866.4**	-54	63.26	13.36	**84.26**	33.74	-13.2	-5.08	**36.6**
6Å Solvent Separation	Planar	Very high detail	FAS	4.4	-179	54.06	-122	**223.2**	-170	49.51	-110	**208.6**	171.6	-50.1	110.9	**210.3**	-7.22	3.979	-11.2	**13.92**	1.612	-0.57	0.547	**1.797**
			ACHE	4.0	-916	-44.9	321.8	**971.9**	-871	-35	306.8	**923.8**	867.1	55.41	-326	**927.9**	-48.9	10.55	-3.77	**50.15**	-3.61	20.44	-18.8	**28.01**
		Recommended	FAS	1.3	-175	54.34	-119	**218.4**	-159	32.96	-94.1	**187.4**	160.6	-33.8	95.01	**189.6**	-14.5	20.57	-23.8	**34.59**	1.892	-0.82	0.869	**2.238**
			ACHE	1.5	-894	-15.6	301	**943**	-808	-13.1	286.7	**857.3**	815.9	9.179	-286	**864.6**	-77.7	-6.46	15.01	**79.35**	8.042	-3.96	0.724	**8.994**
	PNG1	Very high detail	FAS	4.4	-176	51.38	-110	**213.4**	-168	48.08	-105	**204**	170.2	-49.4	106.3	**206.7**	-5.33	1.964	-3.54	**6.697**	1.769	-1.34	1.754	**2.827**
			ACHE	4.0	-876	-51.5	294.2	**925.4**	-846	-45.5	295.7	**896.9**	877	30.63	-297	**926.5**	1.11	-20.8	-3.2	**21.11**	31.44	-14.9	-1.73	**34.81**
		Recommended	FAS	1.3	-176	48.32	-106	**211.1**	-160	32.43	-89.7	**186.4**	162.5	-34.6	91.46	**189.7**	-13.4	13.76	-14.8	**24.25**	2.438	-2.12	1.719	**3.659**
			ACHE	1.5	-877	58.77	297.2	**927.9**	-788	-20.1	278.7	**836.1**	811.6	8.467	-285	**860.1**	-65.4	67.24	12.71	**94.67**	23.55	-11.6	-5.83	**26.9**
"Corrected" Values	Planar	Very high detail	FAS	4.4	3.083	0.442	-0.01	**3.115**	2.899	0.069	-0.14	**2.903**	-1.12	-0.44	0.322	**1.25**	1.96	-0	0.316	**1.985**	1.776	-0.37	0.177	**1.824**
			ACHE	4.0	-1.18	-0.01	0.164	**1.193**	-0.92	0.595	0.535	**1.221**	2.674	-15.9	12.75	**20.54**	1.492	-15.9	12.91	**20.53**	1.752	-15.3	13.28	**20.33**
		Recommended	FAS	1.3	3.313	-0.15	0.089	**3.317**	2.95	-0.35	-0.01	**2.971**	-1.26	-0.16	0.32	**1.308**	2.055	-0.31	0.409	**2.118**	1.692	-0.52	0.312	**1.796**
			ACHE	1.5	-1.94	-0.29	0.044	**1.967**	-1.57	-0.9	0.141	**1.814**	-2.67	1.063	-0.82	**2.994**	-4.62	0.769	-0.78	**4.747**	-4.24	0.158	-0.68	**4.299**
	PNG1	Very high detail	FAS	4.4	3.42	0.212	-0.01	**3.427**	3.506	-0.24	-0.04	**3.515**	-0.82	-0.81	-0.27	**1.189**	2.596	-0.6	-0.28	**2.679**	2.682	-1.06	-0.31	**2.899**
			ACHE	4.0	-2.5	0.633	0.075	**2.577**	-2.68	0.835	0.178	**2.811**	-10	1.355	-1.64	**10.27**	-12.5	1.988	-1.57	**12.79**	-12.7	2.191	-1.47	**12.99**
		Recommended	FAS	1.3	3.041	-0.61	0.181	**3.107**	3.482	-1.05	0.356	**3.655**	-1.04	-0.38	-0.26	**1.136**	2.002	-0.99	-0.08	**2.233**	2.443	-1.43	0.092	**2.831**
			ACHE	1.5	-4.35	2.166	0.159	**4.86**	-3.16	-0.2	0.06	**3.165**	-7.04	1.814	-0.81	**7.312**	-11.4	3.979	-0.65	**12.08**	-10.2	1.615	-0.75	**10.35**

Table 5.4: Intermolecular forces between acetylcholinesterase/fasciculin, as calculated by BEEP. The "corrected" values are where the results for the proteins in isolation (top rows of the table, numbers as found in Table 5.3) are subtracted from the results for the pair of proteins together. This produces the *change* in force components due to the presence of the other protein in each case. Previously (for spheres in Chapter 3) we saw that such "correction" could restore the correct forces between a pair of dielectric bodies, whereas here we see that the results remain dubious.

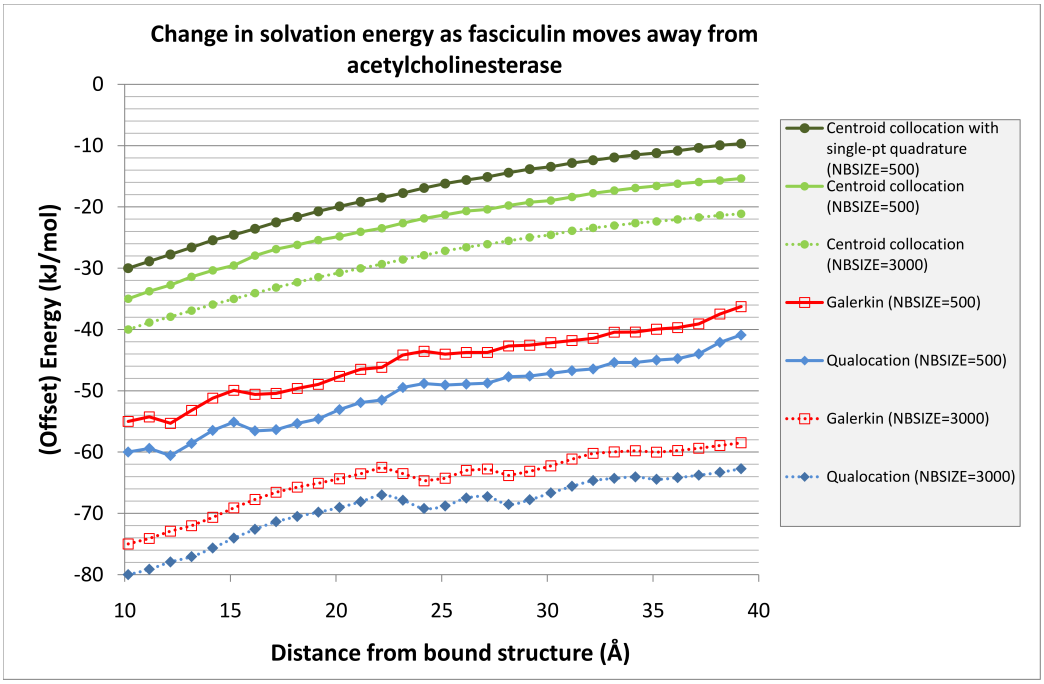


Figure 5.12: Solvation energy for the interaction between acetylcholinesterase and fasciculin at increasing distances (relative to bound structure, PDB 1MAH), for various discretization/integration schemes in BEEP. Please note that these curves have been placed at arbitrary positions on the energy axis to make comparison easier (it is the overall change in energy vs. distance which is of concern, not the absolute values).

fact lie with the "recommended detail" meshes, which give changes in force more comparable to those observed on fasciculin.

PNG1 triangles give (more or less) consistent results between the two mesh representations, whereas the planar triangles do not. The best results seem to be the PNG1-based results for very high detail meshes, which give forces of 2.9 and 12.99 kJ/mol. These are still so far from balancing as to be useless for biomolecular simulation, even though the magnitude of the force on fasciculin does appear to be approximately what we would expect (i.e. consistent with the energy profile in the next section).

5.4.3 Interaction energy for Monte-Carlo simulation

Section 5.4.2 showed that it is not possible to extract useful intermolecular electrostatic forces by direct surface integration of the BEM results. This makes BEM unsuitable for force-based molecular simulation methods such as Brownian Dynamics. However we can compute the total energy of the system as we move the molecules of acetylcholinesterase and fasciculin further apart. Changes in energy as we make small changes to the system can be used as the basis for a Monte-Carlo simulation.

Figure 5.12 shows the total solvation energy as we move fasciculin further from acetylcholinesterase. The values for distance (the x-axis) on Figure 5.12 are the distances along the centre-centre axis from the bound structure, starting from 10.2Å deviation which gives 6Å of solvent between the closest points of the two molecules (as described above). The energies have been re-located on the energy scale (y-axis) in order to allow easy comparison of the curves: the total system energy was around -16,800 kJ/mol, but varied by several hundred kJ/mol for the different discretization/integration schemes, whereas separation dependent changes are close to 20 kJ/mol in all cases.

The energies obtained using the most detailed integration methods appear to be somewhat erratic: there are dips and peaks in the energy plot. In itself this is not too surprising, as the complicated distribution of charges in each molecule (i.e. the distribution of high-order multipoles) could well produce an electrostatic field that leads to harmonic variations in the total energy. On the other hand the two molecules have large net charges (-9 on acetylcholinesterase and +4 on fasciculin), so we might expect the electrostatic behaviour to be dominated by the monopole interaction.

However, changing the neighbourhood size in the FMM octree neighbourhood size results in the peaks and troughs of the curves moving to different places. We conclude that the peaks and troughs are artefacts in the energy profile arising from the discontinuity in integration scheme produced by the near-field/far-field cut-off within the FMM. At a certain point separation, interactions between node-patches become represented by the "crude" FMM (effectively single point qualocation/quadrature) as opposed to by detailed numerical integration. As we slowly move the molecules apart, the structure of the FMM octree results in some near-field patch integrations moving into the far-field, with consequent change in value. We have previously argued that on the length-scales of entire molecules, the net effect on the final solution should be small as the FMM will give very close to the same result as more detailed numerical integration. However the results disprove this: the jagged parts of the energy profile seem to depend on FMM neighbourhood size, not on the numerical integration method, strongly implicating the FMM near-field/far-field cutoff.

If we use a near-field numerical integration scheme which is equivalent to the FMM, i.e. using a single point on each node-patch to represent the integration points, we obtain apparently very smooth results for the energy change as shown by the topmost curve in Figure 5.12. The general shape of the curve looks as though it would match that of the higher-order numerical integration schemes, if the discontinuities in the latter were removed. The correspondence in general shape between the curves (discounting discontinuities) suggests that the effect of the lower accuracy integration on the final result may be small.

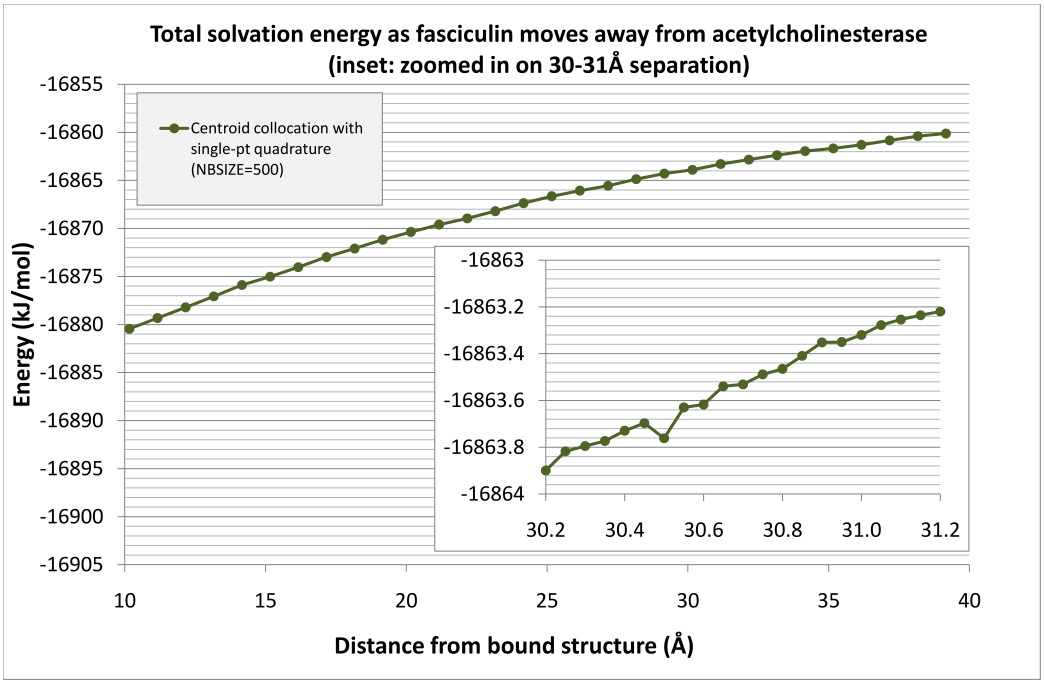


Figure 5.13: Detailed view of the interaction energy in Figure 5.12, between 30Å and 31Å displacement along centre-centre axis (approx. 26Å to 27Å of solvent between closest contacts). The uncertainty or noise in the calculation of changes in energy is on the order of 0.1 kJ/mol.

5.4.4 Force calculation by virtual work

From the energy profile in Figure 5.12 it is possible to calculate approximate forces (along the axis of molecular displacement) by numerical differentiation: i.e. by taking the tangent of the curve at any point.

As a more general principle, it is theoretically possible to apply small displacements to each molecule in the system along each axis in turn, and calculate the force component in each direction as the change in energy divided by the small displacement. This procedure is commonly referred to as "virtual work". The displacement should be small in order that the value for force is approximately constant over the distance being evaluated.

As an example, Figure 5.13 shows a detailed view of part of the energy profile from Figure 5.12. We have calculated energies at a finer scale, which shows that the level of noise in the energy is of the order of 0.1 kJ/mol. Any virtual work force based directly on these noisy energy changes will be highly unreliable. The "force" obtained by taking the gradient of the large-scale curve would be more reliable, but less "accurate" since the displacement over which it is taken is larger. More complicated numerical differentiation schemes exist which could extract a stable and accurate estimate for the force from these energy changes (using multiple points).

In order to calculate molecular forces from the BEM results, each molecule in the system must be moved independently in each of the three directions (plus three rotations if torques are calculated; in a general molecular simulation they are required as well as forces), which is a large number of systems to solve to obtain a set of forces for just one arrangement of molecules in space. We suggest that the extraction of useful forces in this case would be extremely expensive compared to simply using the energies within a Monte-Carlo method.

5.5 Conclusions

BEEP works correctly BEEP is capable of solving the linearised PBE for realistic protein structures and gives results which are consistent with APBS, if some small allowance is made for the effect of differing surface representations and the accessibility of ions. The program works consistently and reliably, as long as detailed and high quality surface meshes are used. For absolute values of solvation energy a detailed numerical discretization/integration scheme like qualocation is recommended. However the users should be aware that the absolute values of solvation energy are very sensitive to the model parameters such as solvent probe radius, and do not necessarily have much meaning in themselves.

For changes in solvation energy of a static system in which no spatial movements are made (e.g. by adding salt) the choice of numerical integration method does not seem to matter very much. For changes in energy where node-patches can move relative to each other, the FMM method can be shown to produce artifacts: a "crude" single-point discretization/integration scheme gives much smoother results.

Curved surfaces reduces the source of errors, but it can be difficult to see the benefit in practice We have shown that the absolute values of solvation energy for proteins are somewhat sensitive to the integration scheme, and require a large number of node-patches to give converged results. The error in solvation energy can be ascribed to a "fidelity error" and a "surface error". We show that the surface error for a spherical mesh can be much improved by the adoption of a curved triangulated surface, using PNG1 triangles as the basis for node-patches rather than planar triangles. Application of PNG1 triangles to real proteins gives less obvious improvement in terms of the solvation energy as a function of mesh resolution, mainly because the fidelity error is the more important factor for real proteins.

Since PNG1 triangles do not significantly increase the computational cost of BEM calculations, and have demonstrable benefit for spherical systems, we suggest that it is worthwhile using them for proteins even though it is difficult to show any clear benefit.

We cannot obtain intermolecular forces directly We have found that it is not possible to derive useful intermolecular forces directly from the BEM results, in contrast to what is claimed in the BEM literature, for example by Lu *et al.* [99, 105]. We suggest that the failure to extract useful forces stems from the problems we highlighted in Chapter 3 for charges approaching a spherical surface: we cannot derive the electric field vector from BEM results with sufficient accuracy to obtain force components which add up reliably. BEM is not suitable for direct-force simulation methods, such as Brownian Dynamics.

Interaction energies for Monte-Carlo simulation We have found that the only practical method for deriving intermolecular interactions is from the "interaction energy", i.e. the relative change in total solvation energy for a system of interacting molecules. We suggest that this energy can be used as the basis for a Monte-Carlo simulation. The accuracy of such energies seems to be limited to around ± 0.1 kJ/mol for the relatively strongly interacting system of acetylcholinesterase and fasciculin, which is well within the kT "acceptance criteria" which would be relevant for biomolecular Monte-Carlo simulations.

The graph of interaction energy vs. molecular displacement exhibits minor artefacts from the FMM near-field/far-field discontinuity in numerical treatment, unless a corresponding low-order discretization/integration scheme is used such that the near-field and far-field contain the same degree of numerical approximations. The inherent "inaccuracy" of such treatment (when used for individual solvation energies) appears to cancel out for the *changes* in energy we are dealing with here, which is somewhat fortuitous. The fact that the individual solvation energy of a molecule is not perfectly represented or converged seemingly does not preclude the possibility that the *change* in that energy due to the presence of another molecule nearby *is* converged and adequately represented. We suggest that this requires some further study.

The requirement to use a low-order integration scheme when calculating changes in energy due to intermolecular effects, due to the previously mentioned FMM artifacts, is advantageous for large-scale simulation purposes (such as a Monte-Carlo simulation) since the near-field integrations are faster to evaluate, leading to potentially faster performance. On the other hand this is somewhat inconvenient in terms of memory usage, since the decreased cost of near-field interactions leads to a larger neighbourhood size, resulting in more memory required to hold the pre-calculated near-field integrations (see Chapter 4 Section 4.8.4). (The benefit of on-the-fly integration using the GPU becomes relatively more attractive; however the limiting factor remains the speed with which the 12-fold BEM/FMM can be solved).

Chapter 6

Discussion

6.1 The BEM for protein electrostatics

This project grew out of an initial goal to create a new method for carrying out very large-scale biomolecular simulations of cellular processes, on the scale of tens of thousands of macromolecules over a range of seconds. Very early in the project we identified intermolecular electrostatic interactions as the most important factors in such a simulation: most (perhaps all) interactions between biomolecules are ultimately electrostatic in nature, and most of these interactions are mediated by water molecules and heavily influenced by the presence of mobile ions. This led us to the implicit solvent model and the Poisson-Boltzmann Equation (PBE), which offers a representation of macromolecules as charges embedded in low dielectric regions, surrounded by a high dielectric solvent containing ions. Although the PBE is a conceptually simple model (compared to full atomic detail) in practice it is not straightforward to solve numerically, and very few analytic solutions exist at all.

We followed the work of Benzhou Lu *et al.* *[103]* in creating a linearised Poisson-Boltzmann Equation solver based on the Boundary Element Method. Although the BEM has been used previously to model protein electrostatics, Benzhou Lu and his collaborators used a new Fast Multipole Method implementation for the screened Coulomb function which allowed them to create an algorithm capable of solving the linearised Poisson-Boltzmann Equation in linear time with respect to number of particles.

6.1.1 BEEP & BEEPp: Features

At the outset of this project Benzhou Lu (Institute of Computational Mathematics, Chinese Academy of Sciences) [1] had not released any working code, so we implemented our own program, called BEEP, based on algorithms in the literature and making use an existing Fortran FMM library (supplied by Jingfang Huang, University of North Carolina, Chapel Hill [112]). In the process of developing BEEP we have made improvements to the BEM and FMM methods, both algorithmic and computational:

- implemented a new curved-triangle based node-patch discretization which substantially improves accuracy of the BEM for spherical surfaces
- implemented an arbitrarily detailed numerical integration scheme based on Gauss-Legendre integration over recursively subdivided triangles
- implemented a range of discretization options: Galerkin, qualocation, centroid collocation. We have shown that the use of Galerkin-style integration is necessary to correctly account for the near-singular BEM integrations over each surface element, but that in general the qualocation method with 4-point Gauss-Legendre quadrature gives a good compromise between accuracy and speed for all other explicit integrations.
- extended the BEM method to allow different dielectric constants to be assigned to the volume within each protein mesh
- experimented with a wide range of mesh-generating techniques in order to generate reliable surfaces from PDB structures: we suggest using either GAMER, or a combination of MSMS and Meshlab.
- we ported the FMM from Fortran to C++ and then parallelised the entire program to make maximum use of the heterogeneous computational resources available to bioscience researchers (multi-cored CPUs in desktop computers, multi-node HPC clusters, GPU acceleration) and used a variety of programming methodologies: OpenMP, Charm++ and OpenCL. The parallel program is called BEEPp.
- a Python module has been developed to provide easy access to BEEP functionality via this high-level scripting language.

[1] At the time of publishing the BEM papers we have referred to throughout this thesis, Benzhou Lu was working at UCSD at La Jolla in the McCammon group. He has since moved to the Institute of Computational Mathematics at the Chinese Academy of Sciences, Beijing.

6.1.2 By-products of BEEP

In addition to the PBE-solving capabilities of BEEP, the underlying FMM is also available as a stand-alone piece of code (in single-threaded, multi-threaded, OpenCL and Charm++ parallel forms). Furthermore we implemented a simple Generalised Born solver which can be used for "quick and simple" electrostatic analysis (though we do not particularly recommend it: the results are in no way comparable to those obtained by solution of the Poisson-Boltzmann Equation).

6.1.3 Parallelisation

BEEP can be run over several compute nodes of a cluster, with the best return for resources being achieved for 8 or 16 compute nodes. OpenMP gives an increase in performance on multi-core machines by a factor of approximately 3× for quad-core nodes. There is, however, no further speed-up for higher numbers of CPU cores on a single machine, presumably due to limitations of memory bandwidth between RAM and CPU, and possible cache-contention between OpenMP threads (in which the data loaded into the cache by each thread is continually overwritten by data for other threads).

The substantial raw computing power of modern GPUs (made accessible to programmers through the GPGPU methodologies like CUDA and OpenCL) allows us to speed up numerical integrations by a factor of several hundred. Unfortunately within the wider context of BEEP the speedup achieved by use of GPU acceleration is limited to a factor of approximately 1.5, as the numerical integrations do not take up much more than a third of the original run-time, even once the shift in optimum neighbourhood is taken into account: the use of a GPU (or indeed multi-core OpenMP) alters the optimum balance between FMM and explicit near-field integrations in the BEM algorithm, in a non-trivial manner.

On a modern GPU workstation with 12GB of RAM, BEEP can solve a meshed system with up to 1.5 million vertices in approximately 30 minutes (depending on the number of iterations required to solve the matrix-vector BEM equation). Making slightly more efficient use of the GPU, the limit to system size is more modest: around 220,000 vertices, or around 340,000 vertices for systems without a GPU to accelerate near-field integrations. Overall performance in all cases depends strongly on the choice of discretization/integration scheme used, with less detailed numerical integration schemes resulting in reduced run-time at some considerable risk to reliability and accuracy of the results.

The performance of BEEP is, probably, too slow for use in a biomolecular simulation in which forces/energies must be evaluated rapidly at each timestep (a thirty-minute wait for the energy of the system is likely to be too long). Our experience with proteins so far leads

us to believe that accurate surface solutions require many thousands of vertices per protein mesh. If we could find a way to reduce the representation of each protein, this would be of extreme benefit.

6.2 Forces from BEM results

We have found that it can be relatively easy to obtain accurate numerical solutions for electrostatic systems for which analytic solutions exist: i.e. those based on individual charges and spherical cavities (such as the Born Ion or a simple Coulomb's Law interaction (with or without ionic screening).

However we have also found that much of the accuracy (i.e. correlation with analytic values) can be destroyed by simply perturbing the vertices of the "spherical" surface (in reality, the surface is almost always a uniformly subdivided icosahedron with vertices projected to unit radius). Even a very small change in each vertex coordinate, breaks the symmetry of the mesh. Numerical errors can then be seen to appear in the method, which without proper care and attention can seriously reduce the accuracy of the results.

In general the solution to BEM problems becomes more difficult as charges approach the dielectric surface, as illustrated by the "off-centre Born Ion" in Chapter 3: the solutions to the BEM approach a singularity as the charge becomes very close to the surface, with the result that many surface points are required in the vicinity of the charge to adequately represent the surface solution.

There is confusion in the literature over the nature of the electrostatic forces which are at work in the continuum solvent model. We have reiterated the explanation of Gilson *et al.* who separated the force components on a protein under the assumptions of the PBE into three parts: the qE interaction of atomic charges with the reaction field; a dielectric boundary force which arises mathematically from the variation in energy across the discontinuous dielectric interface; an ionic boundary pressure.

The qE force can be solved in one of two ways:

- by directly evaluating the integral of electric field over the volumetric charge density
- through conversion of the volumetric force term into a mathematically equivalent surface-integral over the *inside* of the dielectric boundary (using Gauss' Law) which results in the Maxwell Stress Tensor (MST)

There is some variation in the literature over the derivation of expressions for the dielectric boundary force. Most commonly in the BEM literature the dielectric boundary force is combined with the qE force and the sum of the two are found directly by applying the MST

to the *outside* surface of the dielectric boundary. This is mathematically valid, however we suggest that in order to analyze sources of uncertainty and error in the force calculation it is helpful to separate the various terms in order that their relative influence can be examined.

The MST offers an intuitive representation of the electric field at a dielectric boundary as a physical system of stress acting on the surface. In fact this representation of the electrostatic force as a physical stress is entirely non-physical and exists only in the sense that it is mathematically equivalent to the volumetric body force it represents.

Finally there is an ionic boundary pressure which acts normal to the dielectric boundary. The magnitude of the total force this produces on a protein appears to be very small compared to the other force components. The magnitude of the ionic boundary force does not appear to be very much affected by the presence of other molecules in the vicinity, reinforcing the view that this is a very local effect. The primary effect of salt is in altering the magnitude and extent to which the electric field emanating from a protein extends into the surrounding solvent. The effect of salt on BEM solutions between two molecules is to reduce the extent by which surface solutions of one molecule are affected by another: this effect is captured by the terms of the BEM matrix. The direct effect of salt in terms of boundary pressures is very small in comparison to the effect it has within the BEM equations, via the screening constant κ.

In Chapter 3 we showed that for an off-centre Born Ion these individual force components are extremely difficult to find with high accuracy from BEM surface solutions. This stems from our inability to accurately reconstruct the electric field vector at the surface: we generally have the normal component with high accuracy, but must interpolate the planar components (parallel to the dielectric boundary) from the gradient of surface potential which introduces an inevitable numerical error. The net force on the off-centre Born Ion remains non-zero even for relatively high mesh density.

It seems likely that the failure to compute reliable electric field vectors is intrinsic in our choice of constant basis-function over each node-patch[2]. If we could adopt a higher-order basis function we could perhaps obtain values for electric field which give a consistent total force over the molecule (assuming we use methods which do not lead to large numerical error in our summation). Unfortunately it is very difficult to incorporate a linear-element basis function into the FMM, since it is inherently a single-point representation: the mean linear values for BEM quantities at node-patches could be used, but for the far-field this is fundamentally no improvement on the constant-element approach.

[2]i.e. that the values of f and h satisfying the BEM matrix-vector equation are assumed constant over the area of each node-patch.

It is possible that there exists a numerical method which solves this problem, perhaps by smoothing the surface potential prior to numerical differentiation. However we have been unable to find one.

6.3 Difficulties and future work

In Chapter 5 we showed that BEEP can be used to find the electrostatic component of solvation energy for "real" proteins (using high-resolution crystal structures taken from the PDB) with accuracy roughly comparable to another widely available PBE solver (APBS). BEEP and APBS do not give identical results because they represent the dielectric boundary and ion-accessible regions differently, making direct comparison difficult.

In order to capture electrostatic solvation energies which are likely to be reasonably well converged, we recommend a meshing density of around 1.5 vertices/$\mathring{A}^2$. We showed that use of a curved triangular representation of the dielectric boundary could reduce the amount of error inherent in the BEM discretization (both in terms of fidelity of the solution, and geometric correspondence of the mesh to the "real" dielectric surface).

The difficulty we found in calculating accurate forces on a spherical cavity with off-centre charge were even more apparent when we moved to the highly curved surface of proteins, where the net force (i.e. the extent to which the force components did not add up to zero) was substantial.

When two proteins are placed adjacent to one another in space, the value for surface solutions does not vary very much compared to the surface solutions for the proteins in isolation. Consequently, the forces on each molecule are dominated by the erroneous "net force" which can be calculated for the isolated case. Unfortunately, when the "isolated" force is subtracted, sensible net forces for the interaction between the molecules does not result, even for very highly detailed meshes.

On the other hand, the total solvation energy for a pair of interacting proteins (in Chapter 5 we use the example of acetylcholinesterase and fasciculin) is a more converged quantity, which appears to stably and accurately represent the electrostatic interaction between the bodies. In principle one could find a way to calculate forces from these for use in a Brownian Dynamics simulation, however practically we suggest it is not worth the effort. The total energy could be used as the basis for a Monte-Carlo simulation, however as we noted previously the relatively slow performance of BEEP for highly detailed protein surfaces does not make this a particularly viable option for large-scale biomolecular simulations.

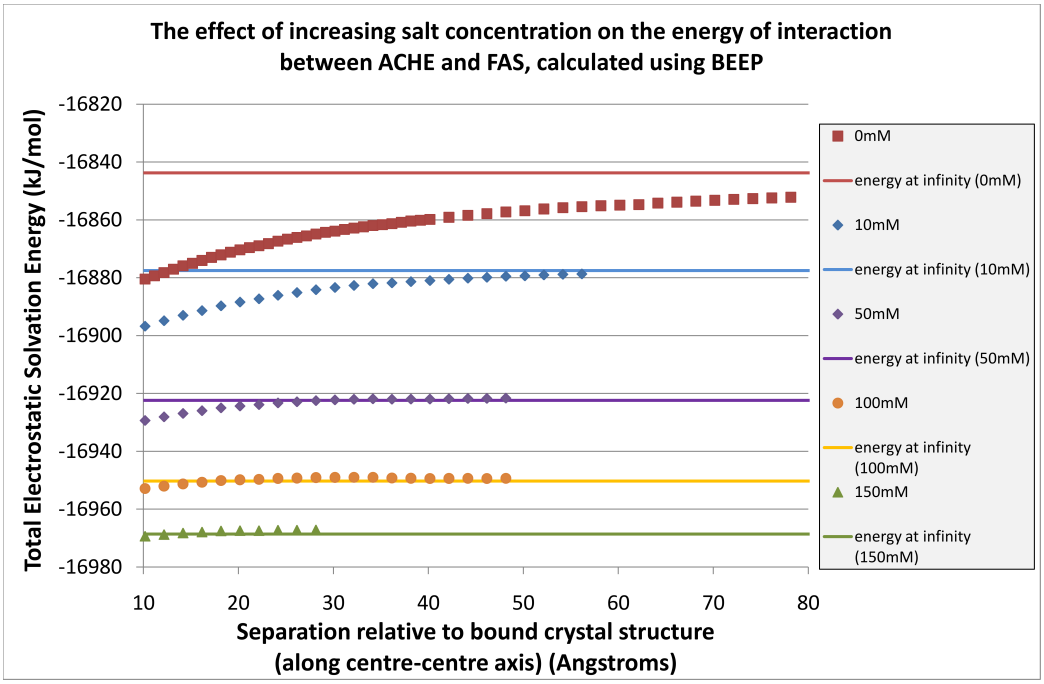


Figure 6.1: The effect of salt on the interaction between acetylcholinesterase and fasciculin

6.3.1 The effect of salt

Figure 6.1 shows the effect of salt on the interaction between acetylcholinesterase (ACHE) and fasciculin (FAS) (as previously described in Section 5.4.2 of Chapter 5). The addition of salt very rapidly eliminates the energetics of the molecular interaction. For all non-zero salt concentrations, the length over which the two molecules "interact" (i.e. the range of distances for which the total energy differs significantly from that calculated for the pair in total isolation from one another) is much shortened compared to the zero-salt case. In addition, the total solvation energy of the system is reduced, which corresponds to the fact that the continuous salt distribution can adopt an energetically favourable configuration which lowers the total energy.

It may seem surprising that at just 100mM salt concentration the interaction between ACHE and FAS is so drastically damped, even though the molecules are physically separated by just 6Å of water (at their closest approach). 100mM salt is much lower than what would be expected for physiological conditions[3], which would suggest that (if this result were physically correct) the mutual electrostatic attraction between these molecules only takes place when they are already close enough to be very close to actually binding, which contradicts the more conventional wisdom of a “long-range” electrostatic steering potential.

[3]In the case of these two molecules, they would naturally occur in the extracellular space of a synapse rather than in a cellular cytosolic medium; in any case we would expect an ionic concentration greater than 100mM.

In fact we are inclined to suspect that whilst BEEP is giving the mathematically correct answer, the continuum solvent model as it is implemented in BEEP is producing a misleading result. (We found in Chapter 3 that BEEP does indeed give the correct form of the screened Coulomb potential for a pair of small ionic cavities separated by ionic solution, so we have some confidence that BEEP is not functioning incorrectly).

The problem is that salt ions in reality represent quite highly charged objects, and will be well solvated with a strong energetic disincentive against sufficient desolvation to approach very close to the surface of a protein. Therefore in reality we suppose that there is a significant layer around the protein where ions are unlikely to be found (both due to the water loosely bound to the surface of the protein, and the "cage" of water molecules solvating ions).

Within BEEP no such layer of water molecules exists, and ions are free to approach to the very dielectric boundary itself, results in a salt distribution which appears to almost completely negate the electric field of the protein, on a lengthscale very close to the protein surface[4].

Zhou [154] has described a theoretical method for describing the interaction between macromolecules and salt under a continuum solvent model. Thomas and Elcock [155] found that this gave consistent results with explicit solvent MD simulations intended to mimic the interactions between salt and highly charged side-chains of glutamate/aspartate and lysine, using a mixture of water, salt, methyl-ammonium and acetate as a model. If correct, these results can be used to provide a more rigorous basis for defining an ion-exclusion layer within the Poisson-Boltzmann model, which we would expect to improve the validity of our calculations for protein interactions in the presence of salt.

6.3.2 Comparison with other methods for biomolecular interaction

In Chapter 1 we introduced the test-charge approximation and the Effective Charges for Macromolecules (ECM) method (as described by Gabdouilline and Wade [88]) for approximating the electrostatic interaction between two objects in solution. These methods were developed to give a quick and simple way to estimate intermolecular forces based on precomputed solutions to the more-rigorous Poisson-Boltzmann Equation for the isolated molecules. ECM is an advance on the test-charge method which scales charges to take into account the effect of low dielectric, and the consequently lowered screening of charge-charge interactions, existing between two proteins in solution.

[4]It might also be noted that in so doing this probably violates the primary assumption of the linearisation of the PBE: that the potential in ion-accessible regions is significantly less than kT/e

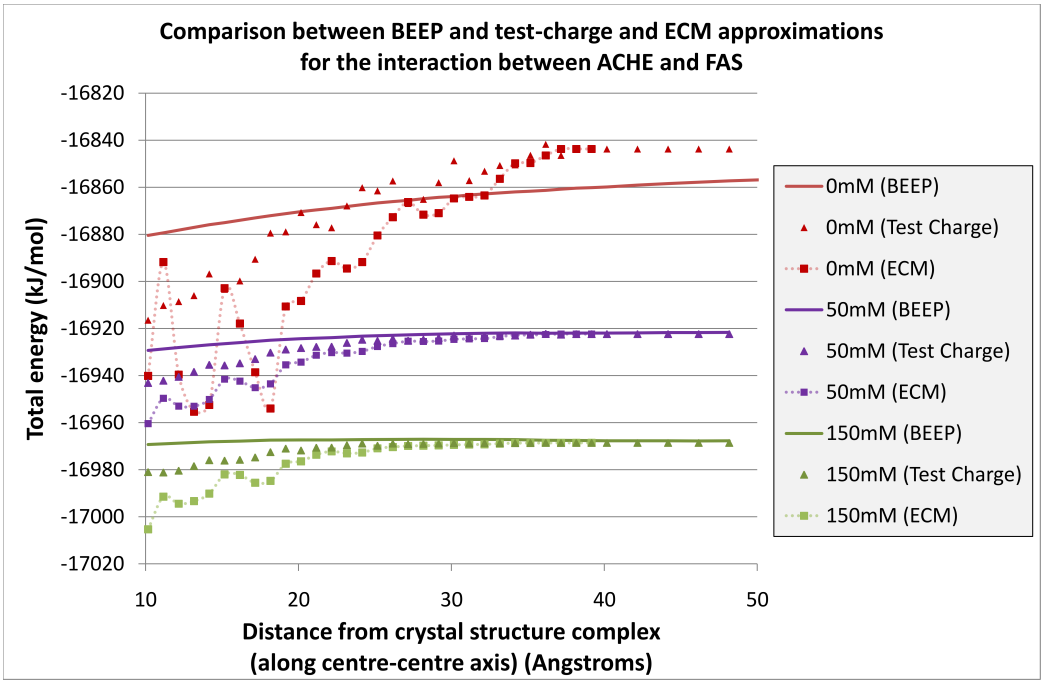


Figure 6.2: Comparison between BEEP and the test-charge and ECM methods.

Figure 6.2 compares our results for the interaction between acetylcholinesterase and fasciculin at varying salt concentrations, starting from a separation (relative to the bound crystal structure, as in PDB 1MAH) of 10.2Å. For the zero salt case, the test-charge and ECM results are highly erratic, reflecting the high potential and rapidly changing field in the volume between the proteins: these results use a rather coarse potential grid (generated using APBS), where we should probably use a more finely detailed grid.

However for non-zero salt, both ECM and test-charge methods suggest that there is some interaction (not so erratic as for zero salt) where BEEP suggests practically none. This makes us inclined to conclude that BEEP has perhaps overestimated the salt effect, as described above.

Experimental Setup The test-charge approximation and ECM method require a grid of potentials for each, on which the test-charges or effective charges of the other molecule are "overlaid" to give the total energetic interaction. We used APBS to solve the PBE for ACHE and FAS (taken from PDB structure 1MAH, as described previously), with protein dielectric of 2.0, solvent dielectric 80.0, varying salt concentrations (0mM, 50mM, 150mM), solvent probe radius 1.5Å, charges and radii assigned using the PARSE force-field; all other settings were default APBS values, as generated by PDB2PQR. For each molecule at each salt concentration, APBS wrote the potentials for a cube of grid points, centred on the centroid

of the charges associated with each molecule, with a side-length of 102Å and uniform spacing of 0.53Å between grid points in each direction.

For ECM effective-charges, the potentials calculated by APBS were used to find a set of effective charges using the ECM program [5], following the exact steps suggested in the "example" subfolder of ECM. The fitting of effective charges to potential was carried out for a "skin" extending from 4Å to 7Å (ACHE) or 5Å to 10Å (FAS)[6] and effective charges were placed at the centres of Cα atoms.

6.3.3 The effect of macromolecular crowding

Cellular conditions are crowded and salty. This means that any pair of interacting proteins will be at all times surrounded by other proteins of various sizes and charge distributions, which will have a number of effects on the reactivity of the two proteins with each other. Firstly each protein needs to diffuse to the other in a restricted space rather than in an unrestricted infinite volume [156]. The reduction in effective diffusion coefficient means that it takes relatively longer for proteins to arrive in the vicinity of one another, and also more time to move apart. The rates of reactions could become diffusion limited under these conditions, which for certain time-critical systems (such as cell-signalling pathways, perhaps) would imply that there must be some minimum concentration (or confinement) of the various proteins in order to guarantee timely responses to stimuli, without having to wait for diffusion to the relevant proteins into range.

The second effect of the crowdant molecules will be as a source of electrostatic "interference": for example, the electrostatic force on fasciculin will be less directed toward acetylcholinesterase if there is a group of other, large, proteins nearby, even if they are not so heavily charged as acetylcholinesterase (since nearby charge distributions will produce a stronger field according to the inverse square law).

Crowdant molecules themselves, apart from the effect of their charge distribution, also represent regions which are substantially ion-excluded with a low dielectric constant. We suggest that the salt-excluding effects of crowdants could have some impact on the mutual interaction between a third-party pair of proteins, though the preliminary results illustrated in Figure 6.4, based on the interaction between ACHE and FAS in the presence of spherical crowdants as illustrated in Figure 6.3 (with dielectric constant $\varepsilon_{crowdant} = 2.0$) do not support this hypothesis.

[5] ECM is included as part of the SDA package (acronym for "simulation of diffusional association of proteins") [52] available from: `http://projects.villa-bosch.de/mcmsoft/sda/4.23/sda.html` (version 4.23).

[6] These are the maximum dimensions of "fitting region" for which we could obtain stable behaviour of the ECM program. The code appears to be around 10 years old: it appears to work correctly (compiled from source by gfortran version 4.5.0), but has hard-coded limits on the size of problem it can manage.

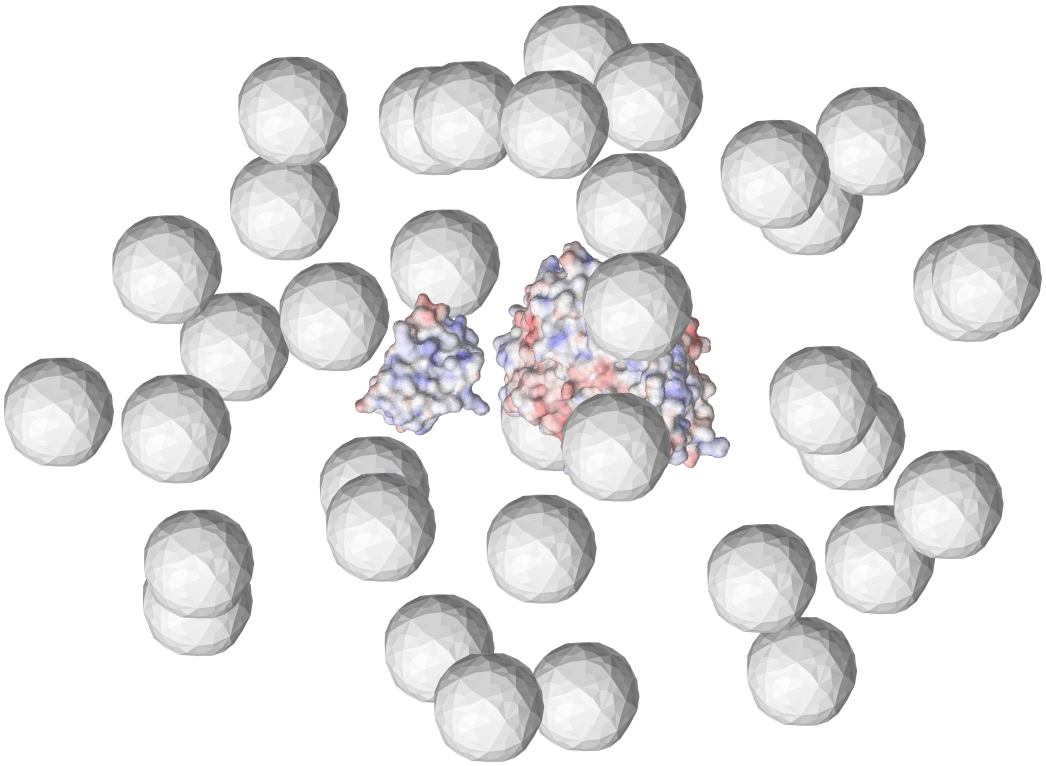


Figure 6.3: Illustration of the spherical "crowdants" around acetylcholinesterase and fasciculin

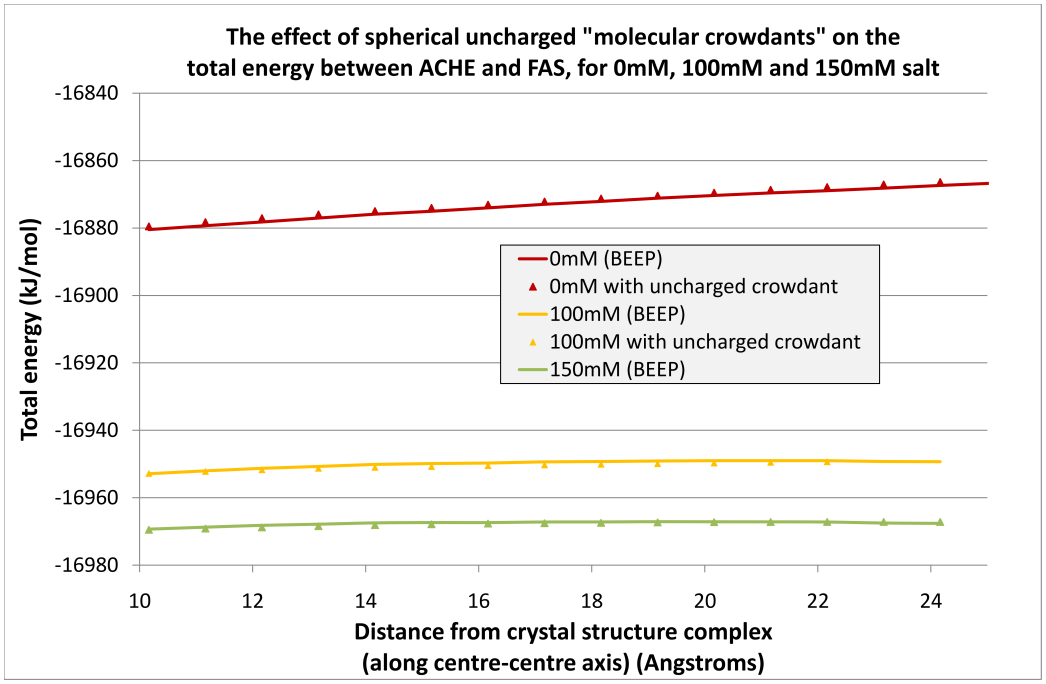


Figure 6.4: Graph showing the effect of molecular crowding on the interaction between acetylcholinesterase and fasciculin at close range, for reasonably low salt concentration. The effect is negligible, and does not really exceed the noise margins implied by Figure 5.13: the expected decrease in electrostatic screening due to the presence of the ion-excluding low-dielectric bodies is not apparent.

6.3.4 Uncertainty in dielectric constant

Modifying our assumptions for the dielectric constants of ACHE and FAS does not significantly alter their interaction, as long as the dielectric constants remain equal between the proteins, as illustrated in Figure 6.5. However if the dielectric constants of the two bodies are not equal we see a surprising effect: allowing fasciculin to undergo greater polarisation (higher dielectric constant) relative to acetylcholinesterase leads to a much increased magnitude of the interaction energy between the two bodies. The effect of a higher dielectric constant of 20.0 applied to ACHE with a dielectric of 2.0 for FAS produces a less dramatic change in energy curve.

We don't know what the dielectric constant is "in reality" for either ACHE or FAS since assigning a single dielectric constant to an entire protein is somewhat dubious in any case. However the total polarisability of the protein including the effect of solvent-exposed sidechains will (we assume) depend somehow on the ratio of solvent-exposed surface area to the volume of the protein (which mostly represents a low dielectric material). Therefore, it seems likely that we could arrive at different values for dielectric constant for ACHE and FAS according to their ratios of surface area to volume: by this rationale FAS would be applied a significantly higher dielectric constant than ACHE.

In quantifying the interaction between ACHE and FAS we have, at least, the level of uncertainty implied by the differing curves in Figure 6.5. It would be of great interest to explore the validity of this effect of changing dielectric constants for the ACHE/FAS interaction by comparison with a more detailed simulation method such as MD. Whilst direct comparison with MD is difficult we suggest that comparison with potential of mean force results obtained by MD (in which perhaps only long-range electrostatic forces are calculated) could be of value here in quantifying the relative polarisability of ACHE and FAS.

6.3.5 Choice of force field

In order to demonstrate the effect of the initial choice of parameterisation (partial charges and radii) on the calculated interaction between molecules, Figure 6.6 shows the energy profile for the interaction between ACHE and FAS for three different widely-used force fields (zero salt, protein dielectric constants of 2.0): PARSE [92] (which we have been using throughout this thesis), AMBER [30] and CHARMM [33] (using the versions of those force field which are included in PDB2PQR [157, 158]).

AMBER and PARSE both give comparable results for the total magnitude of the interaction (relative to the total combined solvation energies for ACHE and FAS in isolation).

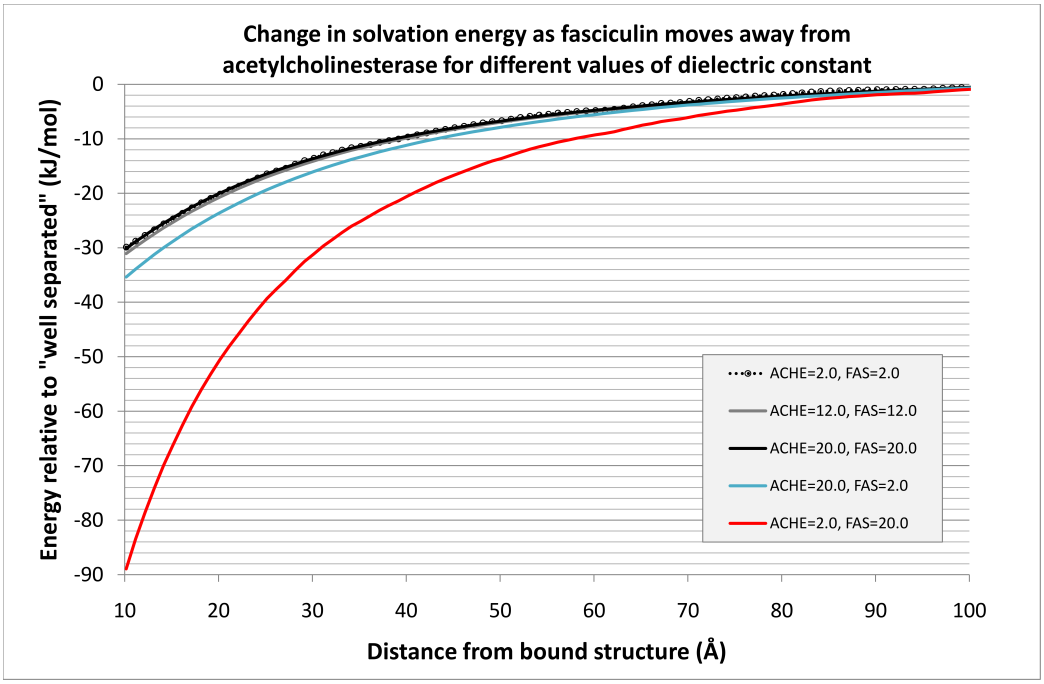


Figure 6.5: Interaction between ACHE and FAS for different protein dielectric constants (solvent dielectric=80, zero salt)

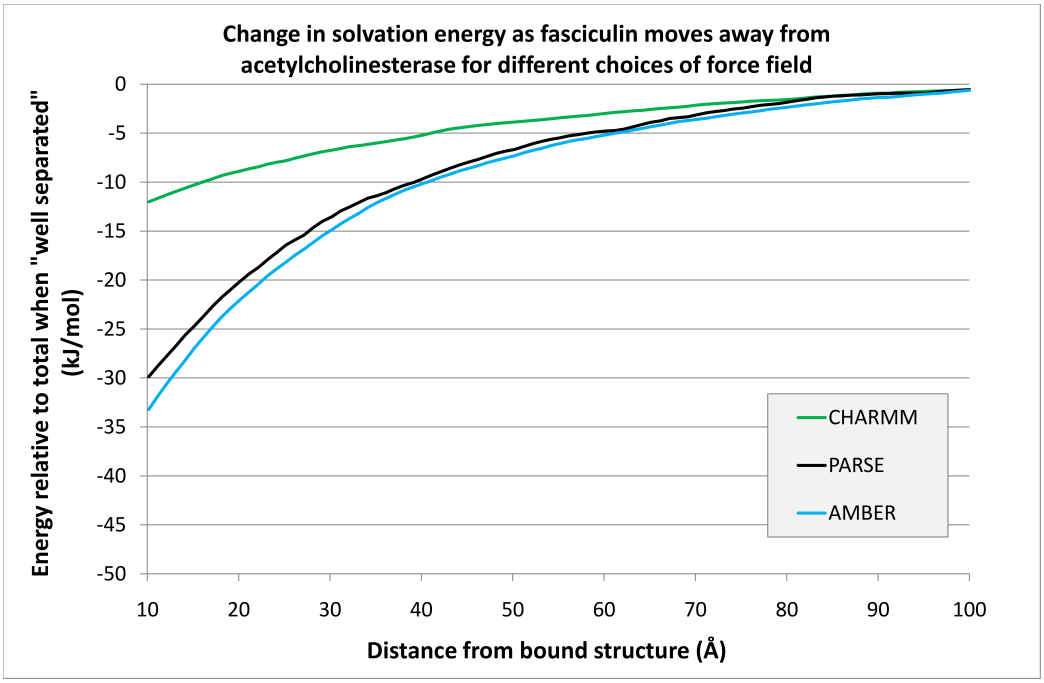


Figure 6.6: The interaction energies for ACHE and FAS (as they are moved apart relative to their bound state as found in the PDB crystal structure 1MAH), calculated using different force fields. AMBER and PARSE show good agreement, whilst CHARMM gives quite different results to both (even though CHARMM and AMBER both produce comparable figures for the total solvation energy, around 4,000 kJ/mol lower than the figures calculated by PARSE).

CHARMM, however, gives results which are much lower, suggesting that the interaction between ACHE and FAS is not so strong.

The absolute values for (total) solvation energy calculated using CHARMM and AMBER differ from one another by only 200 kJ/mol (approximately), but differ from PARSE by around 4,000 kJ/mol. Therefore the difference in the calculated interaction between the two proteins cannot be ascribed to CHARMM underestimating the solvation energy of either of the molecules relative to the other force fields: the difference is entirely due to the different dielectric surfaces and charge distributions produced by the three parameter sets.

6.3.6 Uncertainty in structure

We do not actually know where in space the atomic charges are, and consequently where exactly the dielectric/ionic boundaries lie. In general for electrostatic studies, we use crystal structures obtained from x-ray diffraction or NMR experiments, which result in varying degrees of uncertainty in the precise location of certain atomic groups or flexible chains within molecules. Also of interest are intrinsically disordered regions of proteins, where in solution the structure samples conformational space according to the local environments (which alters the general "energy landscape"). Both of these situations require a flexible representation of protein structures within the continuum solvent model, which BEEP does not currently offer.

6.3.7 Multiple Concentric Boundaries

In order to remove the sensitivity of BEEP to salt concentration, and improve the representation of "dielectric constant" in proteins, we strongly recommend further work to implement more complicated boundaries around each molecule.

In general we propose that it would be better for a protein to have multiple dielectric constants, to represent the differing polarisability of the “core” of the protein (electronic polarisability, $\varepsilon_r \approx 2$), or more freely rotating side-chains at the solvent-exposed surface ($2 \leq \varepsilon_r \leq 20$). A general schematic of the proposed protein model is illustrated in Figure 6.7: the central low dielectric "core" is surrounded by a more “medium-valued” dielectric region (also solvent-excluded and impermeable to ions), several Angstroms in thickness, to represent the outer edges and any bound water molecules. Finally there is the bulk solvent. An additional solvent-dielectric layer which is impermeable to ions could also be added to further extend the Stern layer if necessary.

A similar idea to this has already been proposed by Altman *et al.* [106] who give details for how the surface representation of each region (and it’s continuity with all other surfaces

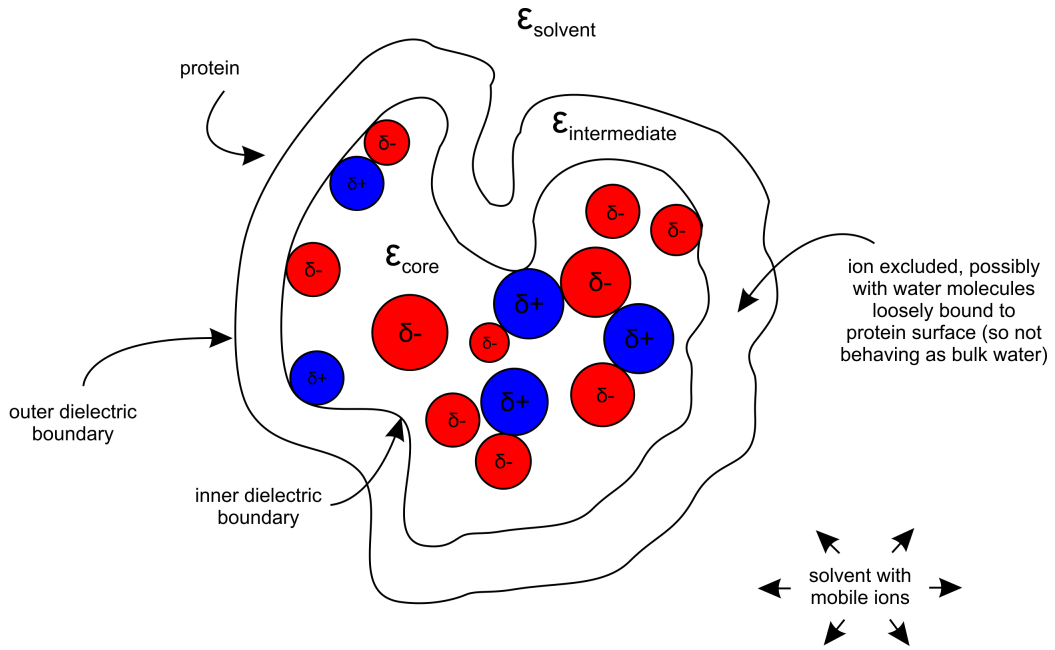


Figure 6.7: Concentric dielectric boundaries for better salt-exclusion and more "realistic" variation in dielectric constant.

within its volumetric boundary, obtained by applying Green's second identity to each region) can be hierarchically ordered into a grand BEM matrix with the correct layout and interactions between elements. Unfortunately the resultant matrix is not directly solvable using the FMM in BEEP, without substantial modifications to the way in which the space is divided.

6.3.8 Simplified Molecular Representation

Alternatively (and more simply) the protein molecules could be represented in some other way, such that the dielectric/ionic boundaries are remote from the charges themselves, and the dielectric constants (or the value of the charges themselves) are in some way rescaled to account for the net effect of this on other entities in the simulation space. The advantage of this is that the total volume occupied by the protein could be altered to take into account the volume swept by flexible parts of the molecule. A picture of what such a "simplified" mesh might look like for acetylcholinesterase and fasciculin is given in Figure 6.8.

From the outset we have been interested primarily in the effect of each molecule on other molecules, not on the details of the solvation energy of the molecule itself. So far we have been forced to represent each protein at very high detail in order to achieve tolerable accuracy in the surface solution for that protein, in isolation. In reality we do not care particularly about the protein in isolation, and it is only the overall effect of the protein

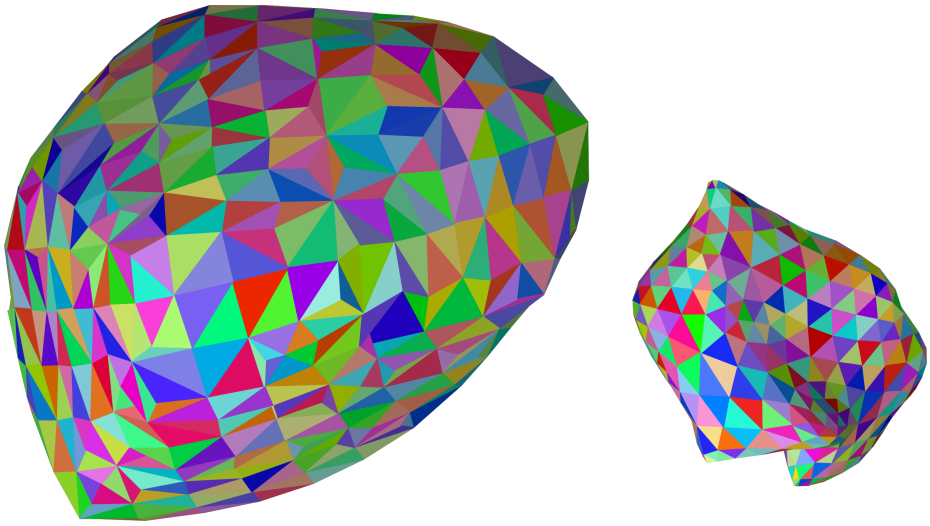


Figure 6.8: Picture of "simplified" triangulated surface representations of acetylcholinesterase and fasciculin: these are not the dielectric surfaces, but smoothed and expanded surfaces which contains all of the original atoms.

on other remote objects that we care about: that is, surface detail is not of interest to us, except insofar as the changes in surface potential/field result in a change in the total energy of each molecule.

We remain optimistic that it is possible to capture the changes in surface solution, and hence total energy, caused by the presence of remote objects without requiring extreme discretization of the surfaces: we would like to radically coarsen the surface, and relocate charges within the molecule such that the overall low order multipole behaviour of the protein at long range is retained, and a suitable polarisation response is induced at medium range (e.g. at a distance of one Debye screening length, or a molecular radius, whichever is greater). At short range we are resigned to the fact that the assumptions of the continuum solvent model break down in any case, and we are not concerned if the short-range accuracy of our own representation of the model is poor: there is no point in accurately finding the solution to a fundamentally incorrect model.

Naturally we would expect the fidelity and accuracy of such a model to be inferior to the highly detailed calculations we have so far obtained from BEEP. However since BEEP currently relies on broad assumptions for salt concentration, dielectric constants, not to mention the extreme sensitivity to surface definition, we would argue that the level of uncertainty inherent in what we are attempting swamps the relative errors we are prepared to accept in a coarse-grained model.

For short range molecular interactions, we suggest that atomically detailed simulations which explicitly take into account the effect of individual water molecules are necessary; the best candidate for such simulations remains MD. How such measurements can be converted

into a "potential of mean force" type parameterisation for use in a larger-scale simulation is yet another problem.

6.3.9 Short range interactions at protein-protein interfaces

The main focus of this thesis has been the discussion of relatively long-range electrostatic forces and their influence on the relative motions of proteins in solution. We have so far avoided the issue of what takes place at short range, but we shall now briefly discuss the effect of short-range electrostatic interactions, specifically the formation of salt-bridges at the protein-protein interface. We have previously introduced the idea of salt-bridges as a type of hydrogen bond in the introductory chapter (Section 1.1.1.2). We described how the stabalizing (or destabalizing) effect of salt-bridges on protein secondary structure could be quantified using theoretical or experimental methods (the latter relating a shift in pK_a to a change in energy).

Xu et al. [159] have previously found that, as a general rule, salt-bridges across protein-protein interfaces are much less "ideal" in terms of geometry than salt-bridges which form during protein folding (i.e. the hydrogen bonds are not linear). This is due to the reduced degrees of freedom available to the proteins to explore conformational space within the binding interface, compared to the flexibility of an unfolded chain. Consequently the salt-bridges in protein-protein interfaces are less energetically favourable than one would usually expect, and water molecules are commonly found to provide "bridging" between polar groups, and mediate the hydrogen-bonding network.

Salari and Chong [160] used both implicit and explicit solvent electrostatic models to demonstrate that salt-bridges in a number of protein-protein interfaces resulted in a desolvation penalty varying between 2.5 kJ/mol and 50 kJ/mol: in these cases, the cost of removing water molecules from around the polar side chains of the protein seems to outweigh significantly the energetic payoff of forming (non-ideal) salt-bridges.

These desolvation penalties are sufficiently large that, at short range, it seems likely that the large effects of a small number of water molecules around exposed polar groups of the proteins will dominate the local energy landscape. In the context of dynamic simulations, an implicit solvent model such as that employed by BEEP is inadequate to represent the subtle energetic contributions of individual hydrogen bonds. We suggest that BEEP could perhaps be extended to model short-range electrostatic effects by some simple metric based on the number of polar atoms with potential to form salt-bridges in a given volume of space. Further work using more detailed electrostatic methods (e.g. using molecular dynamics) would certainly be required to ascertain what functional form this short-range parameterisation should take.

6.3.10 Performance Improvements

The performance of BEEP could be substantially improved by evaluating the FMM on the GPU where possible. This requires arithmetic of double precision complex numbers, which are not currently part of the OpenCL standard. On the other hand it is likely that the CUDA toolkit from NVIDIA will have these features at some point in the near future, and furthermore the support for C++ language features in CUDA is improving rapidly: consequently it seems highly likely that implementing the rate-limiting parts of the FMM on the GPU will become possible, with relatively little effort required, very soon. From our profiling of the BEEP in Chapter 4, we anticipate that this would give a very substantial speedup to BEEP.

Appendix A

Additional Mathematical Formulae & Derivations

A.1 BEM equations for multiple molecules

The derivative BEM formulae for single molecules is given by Equations 2.32 and 2.33 in Chapter 2, repeated here for convenience:

$$\left(\frac{1}{2\varepsilon_{ratio}}+\frac{1}{2}\right)f_p = \overset{PV}{\oint\limits_{\Gamma}}\left[\left(G_{pt}-u_{pt}\right)h_t-\left(\frac{1}{\varepsilon_{ratio}}\frac{\partial G_{pt}}{\partial n}-\frac{\partial u_{pt}}{\partial n}\right)f_t\right]d\Gamma_t + \frac{1}{\varepsilon_{ext}}\sum_k q_k G_{pk}, \qquad p\in\Gamma \quad \text{(A.1)}$$

$$\left(\frac{1}{2}+\frac{1}{2\varepsilon_{ratio}}\right)h_p = \overset{PV}{\oint\limits_{\Gamma}}\left(\frac{\partial G_{pt}}{\partial n_0}-\frac{1}{\varepsilon_{ratio}}\frac{\partial u_{pt}}{\partial n}\right)h_t-\frac{1}{\varepsilon_{ratio}}\left(\frac{\partial^2 G_{pt}}{\partial n_0\partial n}-\frac{\partial^2 u_{pt}}{\partial n_0\partial n}\right)f_t d\Gamma_t + \frac{1}{\varepsilon_{ext}}\sum_k q_k\frac{\partial G_{pk}}{\partial n_0}, \qquad p\in\Gamma \quad \text{(A.2)}$$

This can be extended for a set of J molecules surrounded by homogeneous solvent, following exactly the method of Lu *et al.* [103]. We can write the boundary integral relation for a point p on the i^{th} molecule as a combination of (A.1) and (A.2) (which hold over the i^{th} molecule itself) with the addition of a summation for the remaining integrations, from

point p, over all the other boundaries ($j = 1...J$, $j \neq i$):

$$\left(\frac{1}{2\varepsilon_{ratio}} + \frac{1}{2}\right) f_p = \overset{PV}{\oint_{\Gamma^i}} \left[\left(G_{pt} - u_{pt}\right) h_t - \left(\frac{1}{\varepsilon_{ratio}} \frac{\partial G_{pt}}{\partial n} - \frac{\partial u_{pt}}{\partial n} \right) f_t \right] d\Gamma_t^i$$
$$+ \frac{1}{\varepsilon_{ext}} \sum_k q_k G_{pk} + \sum_{j \neq i} \overset{PV}{\oint_{\Gamma^j}} \left[-u_{pt} h_t + \frac{\partial u_{pt}}{\partial n} f_t \right] d\Gamma_t^j, \qquad p \in \Gamma^i \; i = 1, ...J \quad \text{(A.3)}$$

$$\left(\frac{1}{2} + \frac{1}{2\varepsilon_{ratio}}\right) h_p = \overset{PV}{\oint_{\Gamma^i}} \left[\left(\frac{\partial G_{pt}}{\partial n_0} - \frac{1}{\varepsilon_{ratio}} \frac{\partial u_{pt}}{\partial n} \right) h_t - \frac{1}{\varepsilon_{ratio}} \left(\frac{\partial^2 G_{pt}}{\partial n_0 \partial n} - \frac{\partial^2 u_{pt}}{\partial n_0 \partial n} \right) f_t \right] d\Gamma_t^i$$
$$+ \frac{1}{\varepsilon_{ext}} \sum_k q_k \frac{\partial G_{pk}}{\partial n_0} + \sum_{j \neq i} \overset{PV}{\oint_{\Gamma^j}} \frac{1}{\varepsilon_{ratio}} \left[-\frac{\partial u_{pt}}{\partial n} h_t + \frac{\partial^2 u_{pt}}{\partial n_0 \partial n} f_t \right] d\Gamma_t^j, \qquad p \in \Gamma \; i = 1, ...J$$
$$\text{(A.4)}$$

The extra terms in these equations (for surface integrations carried out over the $J-1$ molecular boundaries on which p is *not* located) are inconvenient in that they do not match the combinations of Green's functions within the brackets of the integral over the i^{th} molecule. However Green's second identity can be applied to any of these $J-1$ molecules, with the point p still residing on molecule i, which results in the following equations:

$$0 = \overset{PV}{\oint_{\Gamma^j}} \left[G_{pt} h_t - \frac{1}{\varepsilon_{ratio}} \frac{\partial G_{pt}}{\partial n} f_t \right] d\Gamma_t^j + \frac{1}{\varepsilon_{ext}} \sum_{k^j} q_{k^j} G_{pk^j}, \qquad p \in \Gamma^i \; i = 1, ...J \quad \text{(A.5)}$$

$$0 = \overset{PV}{\oint_{\Gamma^j}} \left[\frac{\partial G_{pt}}{\partial n_0} h_t - \frac{1}{\varepsilon_{ratio}} \frac{\partial^2 G_{pt}}{\partial n_0 \partial n} f_t \right] d\Gamma_t^j + \frac{1}{\varepsilon_{ext}} \sum_{k^j} q_{k^j} \frac{\partial G_{pk^j}}{\partial n_0}, \qquad p \in \Gamma^i \; i = 1, ...J \quad \text{(A.6)}$$

The addition of (A.5) and (A.6) into (A.3) and (A.4) for each molecular boundary J leads to no overall change in the values of the equations since these extra terms sum to zero. However the extra terms restore the functional form of the equations such that they resemble the boundary integral equations for a single molecular boundary. The final equations are the same as those found in Chapter 2 (Equations 2.34 and 2.35):

$$\left(\frac{1}{2\varepsilon}+\frac{1}{2}\right) f_p = \sum_J \overset{PV}{\oint_{\Gamma^j}} \left[\left(G_{pt}-u_{pt}\right) h_t - \left(\frac{1}{\varepsilon_{ratio}} \frac{\partial G_{pt}}{\partial n} - \frac{\partial u_{pt}}{\partial n} \right) f_t \right] d\Gamma_t + \frac{1}{\varepsilon_{ext}} \sum_j^J \sum_{k^j} q_{k^j} G_{pk^j}, \qquad p \in \Gamma^i \; i = 1, ...J \quad \text{(A.7)}$$

$$\left(\frac{1}{2}+\frac{1}{2\varepsilon}\right) h_p = \sum_J \overset{PV}{\oint_{\Gamma^j}} \left[\left(\frac{\partial G_{pt}}{\partial n_0} - \frac{1}{\varepsilon_{ratio}} \frac{\partial u_{pt}}{\partial n} \right) h_t - \frac{1}{\varepsilon_{ratio}} \left(\frac{\partial^2 G_{pt}}{\partial n_0 \partial n} - \frac{\partial^2 u_{pt}}{\partial n_0 \partial n} \right) f_t \right] d\Gamma_t + \frac{1}{\varepsilon_{ext}} \sum_j^J \sum_{k^j} q_{k^j} \frac{\partial G_{pk^j}}{\partial n_0}, \qquad p \in \Gamma^i \; i = 1, ...J \quad \text{(A.8)}$$

A.2 Higher derivatives of multipole expansions

There is a recursive relationship between increasing derivatives of spherical harmonic functions, such that, given a local expansion $L^{\kappa}_{n-1,m-1}$ describing the potential within a spherical region due to a well-separated group of charges under the Yukawa potential (with screening parameter κ) by Equation 2.48 in Chapter 2:

$$\phi(\mathbf{x}) \approx \sum_{n=0}^{p} \sum_{m=-n}^{n} L_n^m k_j(\kappa r) \cdot Y_n^m(\theta, \psi)$$

we can write express the gradient of $L^{\kappa}_{n,m}$ as [103]:

$$\nabla L^{\kappa}_{n,m} = \frac{-\kappa}{2(2n+1)} \begin{bmatrix} 1 & 0 & -1 \\ i & 0 & i \\ 0 & -2 & 0 \end{bmatrix} \cdot \Lambda$$

$$\Lambda = \begin{pmatrix} (n+m-1)(n+m) L^{\kappa}_{n-1,m-1} \\ (n+m) L^{\kappa}_{n-1,m} \\ L^{\kappa}_{n-1,m+1} \end{pmatrix} \cdot \frac{1}{s} - \begin{pmatrix} (n-m+1)(n-m+2) L^{\kappa}_{n+1,m-1} \\ -(n-m+1) L^{\kappa}_{n+1,m} \\ L^{\kappa}_{n+1,m+1} \end{pmatrix} .s$$

where s is the overflow/underflow scaling mentioned in Section 2.3.2 of Chapter 2 to avoid numerical instability when evaluating special functions, and $i = \sqrt{-1}$. Entries for

which the magnitude of m exceeds n, or n is negative, are replaced by zero.

A.3 Kirkwood's formulae for "self energy" of an off-centre charge in spherical cavity

These formulae are taken from Kirkwood [139] and Hill [140] and describe the "self energy" at the location of an off-centre charge (located R_c from the centre) of magnitude Q in a spherical cavity of radius a with relative permittivity (dielectric constant) ε_r (neglecting the singularity in energy represented by the point charge itself), a ratio between external and internal dielectric constants of ε_{ratio}, and Debye screening parameter κ:

$$G_{self} = \frac{Q^2}{8\pi\varepsilon_0\varepsilon_r} \cdot \sum_{n=0}^{\infty} [B_n + C_n] \tag{A.9}$$

where:

$$B_n = \frac{Y^{2n}}{a}\left(\frac{(n+1)(1-\varepsilon_{ratio})}{[(n+1)\varepsilon_{ratio}+n]}\right) \tag{A.10}$$

$$C_n = -\frac{Y^{2n}}{a}\left(\frac{(2n+1)(\varepsilon_{ratio}xq_n(x))}{[(n+1)\varepsilon_{ratio}+n][(n+1)\varepsilon_{ratio}+n+\varepsilon_{ratio}xq_n(x)]}\right) \tag{A.11}$$

$$Y = R_c/a \tag{A.12}$$

$$x = \kappa a \tag{A.13}$$

$$q_n(x) = 1 - \frac{K_n'(x)}{K_n(x)} \tag{A.14}$$

and $K_n(x)$ is the modified Bessel function of the second kind, which has the following properties:

$$K_n(x) = \sum_{s=0}^{n} \frac{2^s n!(2n-s)!}{s!(2n)!(n-s)!}x^s \tag{A.15}$$

$$(2n+1+x)K_n(x) - xK_n'(x) = (2n+1)K_{n+1}(x) \tag{A.16}$$

$$K_{n+1}(x) - K_n(x) = \frac{x^2K_{n-1}(x)}{(2n+1)(2n-1)} \tag{A.17}$$

$$\frac{K_n'(x)}{K_n(x)} = -\frac{(2n+1)K_{n+1}(x)}{xK_n(x)} + \frac{(2n+1+x)}{x} \tag{A.18}$$

The term C_n is zero when there is no salt in the solvent region ($\kappa = 0$).

A.4 Integration of dielectric boundary force

Gilson *et al.* [136] show that the dielectric boundary force on a molecule (volume Ω bounded by surface Γ) can be written in terms of a volumetric force density, which leads to the relation:

$$f_{dbf} = \int_{\Omega} -\frac{1}{2} E^2 \nabla \varepsilon d\Omega \tag{A.19}$$

Since the gradient of dielectric $\nabla\varepsilon$ is zero everywhere across the boundary, the volume integral takes place only over the dielectric interface, which we will assume has a small constant thickness δ, leading to a constant gradient of dielectric given by:

$$\nabla \varepsilon = \frac{\varepsilon_{ext} - \varepsilon_{int}}{\delta} n$$

If we assume that the electric field varies linearly within the dielectric boundary, making a transition from E_{int} to E_{ext}, with value E_p at points inside the dielectric boundary ($p = 0 \rightarrow \delta$) we can rewrite (A.19) as:

$$f_{dbf} = -\frac{1}{2} \left(\frac{\varepsilon_{ext} - \varepsilon_{int}}{\delta} \right) \int_{\Gamma}^{0} \int_{0}^{\delta} \left(E_{int} \cdot E_p \right) dp.ndA \tag{A.20}$$

It is clear from this expression that the dielectric boundary force acts normal to the surface, so we can redefine our goal as the search for the dielectric boundary pressure on a small increment of surface, which allows us to consider only the integral in p. Rotating the coordinate system such that the z-direction coincides with the normal vector of the incremental surface point allows us to assume for convenience that the normal component of electric field E is entirely contained in the z-coordinate. Therefore we can write an expression for E_p in which only the normal component (z) changes:

$$E_p = \begin{pmatrix} x_{int} \\ y_{int} \\ z_{int} + p\Delta z \end{pmatrix}, \; p = 0 \rightarrow \delta \tag{A.21}$$

where x_{int}, y_{int}, and z_{int} are the components of the internal electric field vector E_{int}, and Δz is the rate of change of normal component of electric field over the boundary layer, given by:

$$\Delta z = \frac{z_0 - z_i}{\delta} \tag{A.22}$$

Substituting (A.21) in (A.20) and integrating leads to

$$\begin{aligned} df_{dbf} &= -\frac{1}{2}\left(\frac{\varepsilon_{ext} - \varepsilon_{int}}{\delta}\right) \left| p(x_{int}^2 + y_{int}^2 + z_{int}^2 + z_{int} p \Delta z \right|_0^\delta \\ &= -\frac{1}{2}\left(\varepsilon_{ext} - \varepsilon_{int}\right)\left(E_{int}.E_{int} + z_{int}(z_{ext} - z_{int})\right) \\ &= -\frac{1}{2}\left(\varepsilon_{ext} - \varepsilon_{int}\right)\left(E_{int}.E_{ext}\right) \end{aligned} \tag{A.23}$$

This result is the same as the difference in value obtained by evaluating the Maxwell Stress Tensor on the outside and inside of the molecular boundary. Equation A.23 does not depend on the thickness of the dielectric boundary itself. However this result is only true if the variation in value of dielectric is linear, and only if the value of E^2 within the dielectric (Equation A.19) has the value $E_{int} \cdot E_p$, rather than the more obvious formulation, $E_p \cdot E_p$ (which leads to a term in the final formula related to the square of the normal component of field: this breaks the correspondence with the result obtained by the Maxwell Stress Tensor.

A.5 Derivation of the Maxwell Stress Tensor

Our derivation of the Maxwell Stress Tensor follows Ferraro [138], Chapter 8 pp 229, using Cartesian tensor notation:

Gauss' Law states:

$$\nabla \cdot E = \frac{\rho}{\varepsilon} \rightarrow \rho = \varepsilon \nabla \cdot E$$

Force per unit volume is:

$$F = \rho E$$

as a tensor (with implied summation over the index $i = 1, 2, 3$ andρ representing a dyadic charge density, i.e. mapping vector of electric field into a vector of force, not necessarily acting in the same direction):

$$F_i = \rho E_i$$

Combining force per unit volume with Gauss' Law:

$$\rho = \varepsilon \nabla E = \varepsilon \frac{\partial E_j}{\partial x_j} \tag{A.24}$$

so (substituting for ρ and using the product rule to rewrite the multiplication of E_i and $\frac{\partial E_j}{\partial x_j}$ in an alternative way):

$$F_i = \varepsilon E_i \frac{\partial E_j}{\partial x_j} = \varepsilon \frac{\partial \left(E_i E_j\right)}{\partial x_j} - \varepsilon \frac{\partial E_i}{\partial x_j} E_j$$

The force vector F is composed of the implicit summation over $i, j = 1, 2, 3$.

Curl of electric field is zero (in absence of changing magnetic field, as $E = -\nabla \phi$ (i.e gradient of a scalar field) and vector calculus says that the curl of the gradient of a scalar field is zero: $\nabla \times E = 0$, so in tensor notation (with $i, j = 1, 2, 3$) the following condition holds:

$$\frac{\partial E_i}{\partial x_j} - \frac{\partial E_j}{\partial x_i} = 0$$

so:

$$F_i \quad = \quad \varepsilon \frac{\partial \left(E_i E_j\right)}{\partial x_j} - \varepsilon \frac{\partial E_j}{\partial x_i} E_j$$

expanding the sum over i and j would lead to the cancellation of all the terms in $\varepsilon \frac{\partial E_j}{\partial x_i} E_j$ (by the zero curl property), except for the values where $i = j$ which remain in the sum. So we can re-write F_i (using the kronecker delta δ_{ij}) as:

$$F_i = \varepsilon \frac{\partial \left(E_i E_j\right)}{\partial x_j} - \varepsilon \frac{1}{2} \frac{\partial \left(E^2\right)}{\partial x_j} \delta_{ij}$$

$$F_i = \varepsilon \frac{\partial}{\partial x_j} \left(E_i E_j - \frac{1}{2} E^2 \delta_{ij} \right)$$

so the force per unit volume arises from the divergence of a tensor T such that we can write the total force on a volume by integrating:

$$force = \int F dV = \int \left(\nabla \cdot T\right) dV$$

where T_{ij} is the Maxwell Stress Tensor:

$$T_{ij} = \varepsilon \left(E_i E_j - \frac{1}{2} E^2 \delta_{ij} \right)$$

or, as a matrix:

$$T = \varepsilon \left\{ \begin{array}{ccc} \left(E_x E_x - \frac{1}{2}E^2\right) & E_x E_y & E_x E_z \\ E_y E_x & \left(E_y E_y - \frac{1}{2}E^2\right) & E_y E_z \\ E_z E_x & E_z E_y & \left(E_z E_z - \frac{1}{2}E^2\right) \end{array} \right\} \tag{A.25}$$

A.6 The Maxwell System of Stresses

The stress-system described by the MST gives the correct results for how forces on electrical bodies are related to their electromagnetic environment, in terms of tension along and pressure normal to "tubes" surrounding the lines of electric force. Following the explanation of Ferraro [138] : looking at the MST components in Equation A.25, if we choose the x-axis to be parallel to a line of force (i.e. electric field vector) then $E_x = E$,$E_y = E_z = 0$, and the tensor reduces to a diagonal matrix:

$$\varepsilon \left\{ \begin{array}{ccc} \frac{1}{2}E^2 & 0 & 0 \\ 0 & -\frac{1}{2}E^2 & 0 \\ 0 & 0 & -\frac{1}{2}E^2 \end{array} \right\} \tag{A.26}$$

This gives the principal stresses as being a compression along the x-axis (that is, a compressive stress normal to the plane cutting the electric field line in a perpendicular plane) and an outward pressure acting on planes of stress tangential to the field line. The magnitude of the principal stresses is equal to the energy density per unit area.

Unfortunately, this interpretation of electrostatic forces as mechanical stresses is not consistent with the real behaviour of elastic materials, and certainly not a physically reasonable picture of the case where the exterior dielectric is a vacuum. The Maxwellian system of stress was devised when it was assumed that there was an all-pervading "aether" capable of providing the mechanical stresses described here. Mathematically the two representations are equivalent, but the volumetric "long-range" nature of the electrostatic interaction simply cannot be reinterpreted as a material stress. Ferraro states "we can regard Maxwell's stress system as a convenient mathematical artifice and not physical reality" [138].

Appendix B

BEEP: input files; file formats; meshing; code acknowledgements

B.1 Input files

For maximum simplicity in parsing input configurations BEEP requires input files to be formatted in a certain way: an xml format configuration file decribing the layout of macromolecules in space and the physical parameters (e.g. dielectric constants of the system, salt concentration for the solvent), plus a number of special format input files (which we name mtz files) which contain the description of the charges and meshed surfaces of the macromolecules. BEEP then solves the linearised PBE for the system, and writes the electrostatic potentials and normal components of electric field to a file.

The force and energy terms on each macromolecular object can be calculated from the potential and field components.

The special "mtz" input files (mtz = mesh-tar-zip) are a gzipped tar-format archive file containing mesh data. The files comprising the mesh data are:

1. xyzqr file: the location, magnitude and radius of all charges in the protein in a whitespace separated text file as illustrated in Figure B.1. We provide a script for converting PQR files to xyzqr format.

2. GTS/OFF format mesh file: the triangulated surface mesh, in either the GNU Triangulated Surface (GTS) format or the Geomview Object File Format (OFF), as illustrated for a simple tetrahedron in Figure B.2. In either case, the mesh file contains the vertices of the mesh, and the connectivity of the vertices forming a triangulated

```
# lines starting with # are comments
# xyzqr format: xyz position of charge in first 3 columns (units: Angstroms)
# charge (q) in 4th column (units: e)
# radius of charge in 5th column (units: Angstroms)
-2.774 95.984 2.675 -0.320000 2.000000
-4.05 95.601 3.323 0.330000 2.000000
-4.421 96.424 4.568 0.550000 1.700000
-5.492 96.281 5.151 -0.550000 1.400000
-4.025 94.115 3.69 0.000000 2.000000
-4.887 93.286 2.732 -0.000000 2.000000
-6.352 93.74 2.747 0.100000 1.700000
-7.003 93.532 3.793 -0.550000 1.400000
-6.749 94.377 1.747 -0.550000 1.400000
-2.734 96.979 2.592 0.330000 0.000000
-2.012 95.661 3.235 0.330000 0.000000
-4.867 92.356 3.026 0.000000 0.000000
-4.401 94.018 4.581 0.000000 0.000000
#
# etc.
#
```

Figure B.1: Example of xyzqr file format.

surface. Triangles are defined by a set of three vertices (OFF) or edges (GTS), with an anti-clockwise ordering from which can be calculated the outward-pointing normal for the triangle.

3. coordinate file: giving the xyz coordinates of the centre of the molecule (in local coordinate frame for the macromolecule). The centre of the molecule is used as the reference point for placing copies of the macromolecule in space in some layout - the universe coordinate of a given molecule locates this centre point, relative to which the position of the charges and mesh points are calculated; rotations of the molecule are also applied relative to this, with the rotation quaternion being the rotation applied to the reference mesh in order to rotate it *into* the universe coordinate frame.

4. an xml file: named definition.xml which details the filenames of the above three files. (see Figure B.3 for an example)

Python scripts are included with BEEP to help the user build the input files - i.e. to create the configuration xml file for a layout of macromolecules, and to build the mtz input files starting from a PDB file. The typical process is shown schematically in Figure B.4.

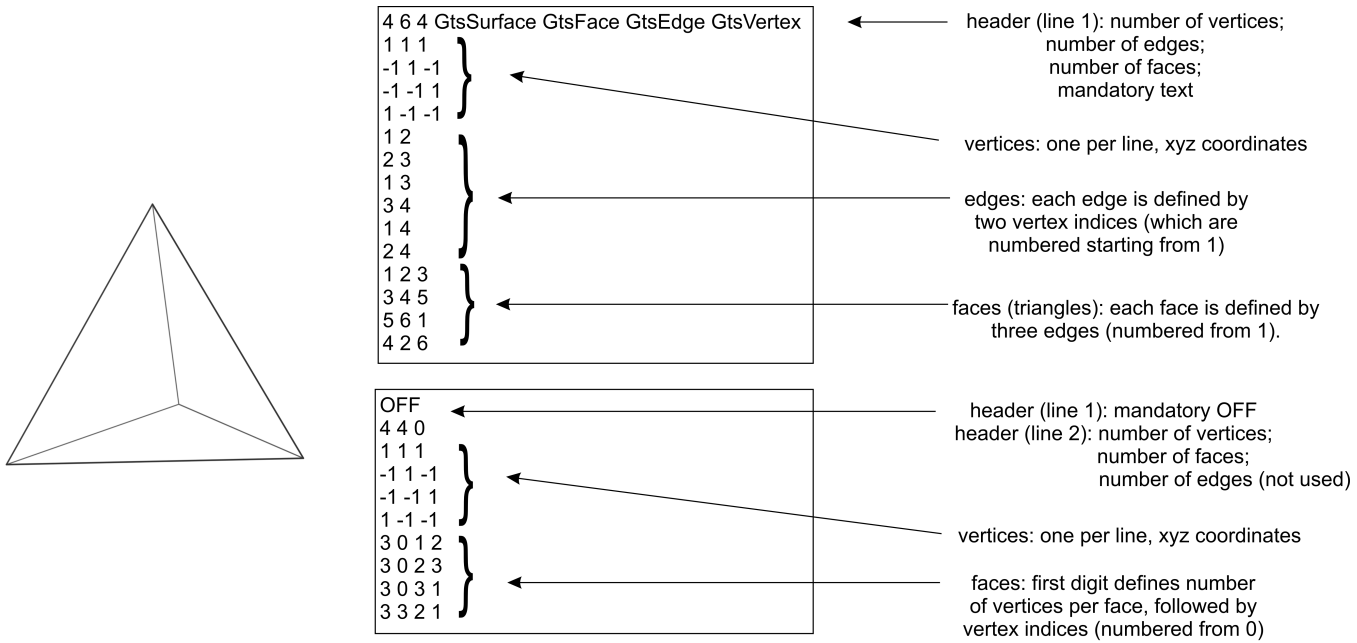


Figure B.2: Example of the GTS (upper) and OFF (lower) file formats for defining a triangulated surface mesh (in this case, a tetrahedron).

```
<?xml version="1.0" ?>
<!-- This is a comment -->
<!-- The BEM_Config tag is the main root element. -->
<BEM_Config>

    <!-- The following xml tags are required to configure BEEP -->

    <!-- The solvent section contains the solvent parameters -->
    <!-- i.e. solvent dielectric and inverse screening length -->
    <solvent dielectric=80.0 kappa=0.102 />

    <!-- This is where we define all meshes we will use -->
    <library>
        <mesh id=0>1MAH-ache.mtz</mesh>
        <mesh id=1>1MAH-fas.mtz</mesh>
    </library>

    <!-- This is the layout of objects in space -->
    <!-- the mesh_id refers to the <mesh> definitions above -->
    <!-- Note that the protein dielectric for each object is defined here -->
    <layout>
         <instance dielectric=2.0 instance_id=0 mesh_id=0 conformation=0 x=0 y=0 z=0 a=0 b=0 c=0 d=0 />
         <instance dielectric=2.0 instance_id=1 mesh_id=0 conformation=0 x=0 y=0 z=0 a=0 b=0 c=0 d=0 />
    </layout>

    <!-- This is the name of the output file -->
    <output>ache_fas_pairwise.fh</output>

</BEM_Config>
```

Figure B.3: Example of the xml file format used to define the layout of objects in BEEP.

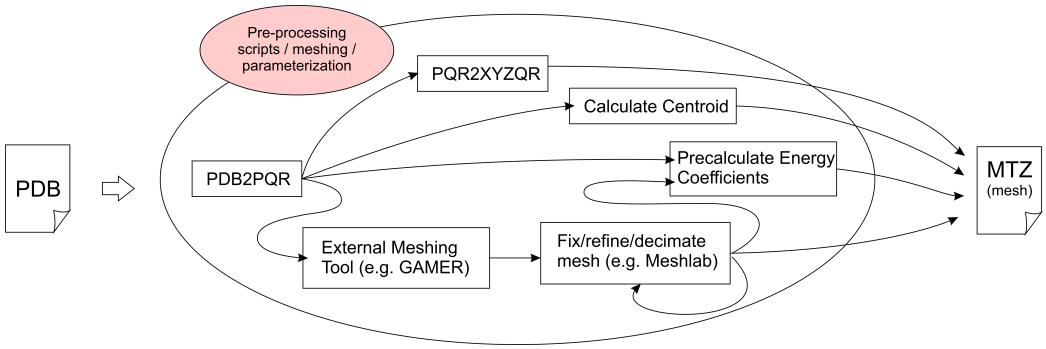


Figure B.4: Schematic diagram for the pre-processing of a PDB file into a BEEP-ready .mtz (Mesh-Tar-Zip) file

B.2 Parameterization

This process requires the use of a third-party tool called PDB2PQR (which must be downloaded and installed separately from BEEP) which is developed and maintained by Todd Dolinsky *et al. [157, 158].* PDB2PQR is the first and most important steps in parameterizing the implicit solvent model. The full details of PDB2PQR are available from `http://sourceforge.net/projects/pdb2pqr/` and from the PDB2PQR documentation, and here we do not go into details except to say that PDB2PQR converts PDB files to the PDB-like PQR format, in which the last two columnsof hte PDB file hold the charge and radius of the charge (instead of occupancy and temperature factor). The charges and radii are chosen according to a molecular mechanics force field, and PDB2PQR offers a number of options here including Amber [30], Charmm [33], and PARSE [92], the latter being a forcefield specifically intended for use in implicit solvation simulations . The protonation states of titratable residues is also determined by PDB2PQR via the PROPKA [161] program (distributed with PDB2PQR), so non-neutral pH values can also be chosen by the user if desired (though the scripts included with BEEP assume a pH of 7 by default). The PQR format file are converted to the "xyzqr" format for the mtz file using a very simple script. The centre of the molecule is calculated as the geoemetric centre of the point charges (the user can re-define the centre of the molecule if they wish).

B.3 Meshing

Once the charge radii are known it is possible to build a solvent-accessible surface for the protein by rolling a "probe" sphere of radius corresponding to a water molecule around the charges. There are a number of choices for protein surface (e.g. Conolly, Richards, inflated van der Waals surface) all of which also depend on the radius chosen for the water molecule. By default we assume a radius of 1.5Å, however a water molecule radius of 1.4Å is also common in the literature.

Programs which can provide a high-quality triangulated solvent-accessible surface are surprisingly rare: MSMS is one well-known program which provides triangulated surfaces however the meshes are commonly of low quality (e.g. have holes, are "non-manifold"[1], have self intersecting faces, regions of extremely high curvature, a high proportion of slender triangles) which we found cause large problems when we try to integrate the BEM kernels over the surface. Consequently some considerable time was spent searching for a mesh-generation/mesh-cleaning regime which can produce reliably high quality surfaces for input

[1] every edge should be associated with exactly two triangular faces

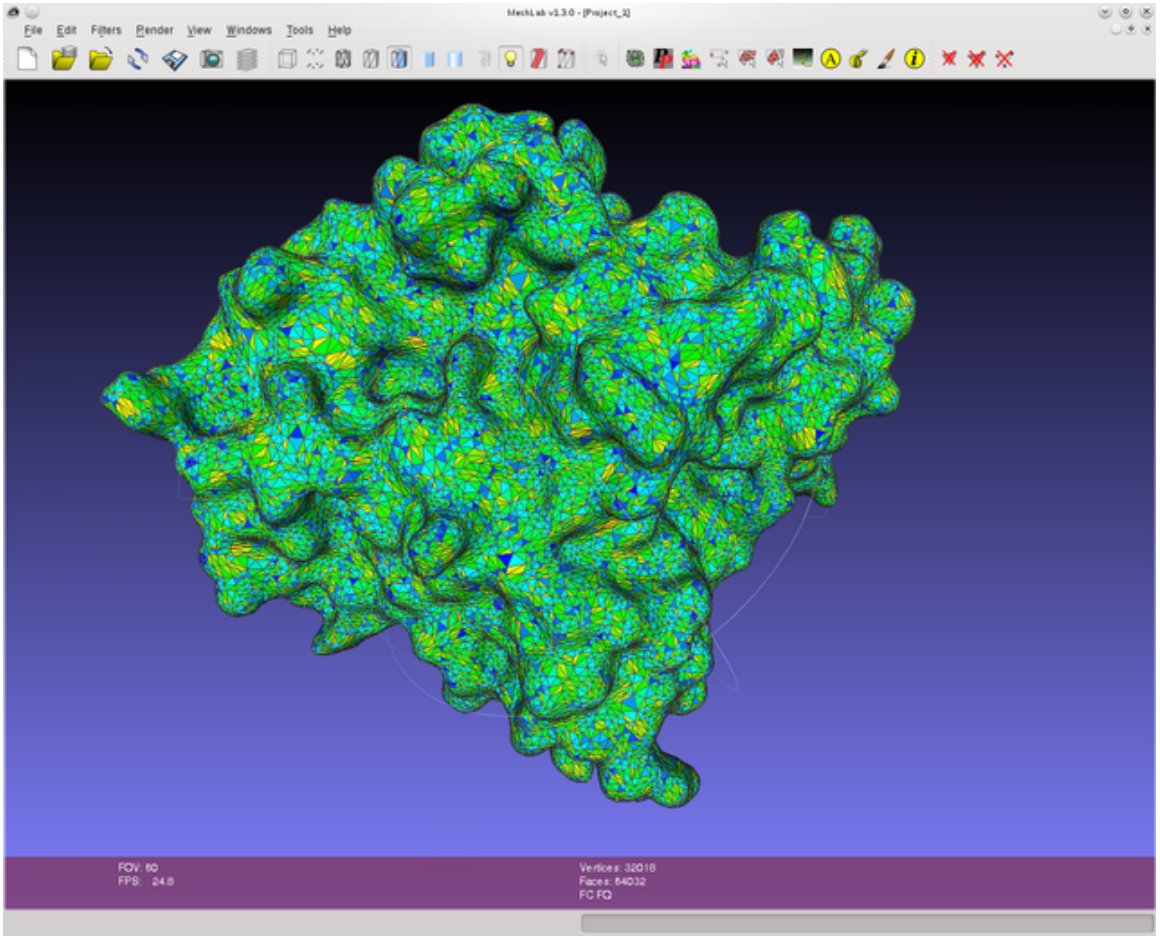


Figure B.5: A screenshot of Meshlab in action: a detailed mesh of acetylcholinesterase (PDB code 1MAH) has been coloured according to triangle quality.

to BEEP. One third-party tool we found particularly useful was Meshlab which is extremely advanced and offers a large palette of mesh-generation features. A screenshot of a mesh within the Meshlab GUI is given in Figure B.5 (coloured according to one of the triangle quality metrics available in Meshlab).

BEEP includes a few simple Meshlab scripts to reprocess MSMS files, and to carry out mesh refinement or decimation to improve/degrade meshes[2]. On the downside, the GTS support in Meshlab is slightly buggy and requires a manual fix to the source code to work correctly with BEEP. Figure B.6 illustrates the conversion of a low-quality MSMS mesh into a much improved mesh suitable for input to BEEP.

We found the most reliable method was to entirely discard the MSMS face data (Figure B.6(a)) resulting in a point cloud of surface vertices (Figure B.6(b)). The Poisson-

[2]This is of interest in Chapter 5 where we look at the minimum meshing detail necessary for accurate BEM results: since the method scales with number of vertices, clearly it is desirable to minimise the number of vertices used to model each object.

reconstruction algorithm in Meshlab can reliably convert this into a topologically valid mesh[3] (Figure B.6(c)). Unfortunately the resultant mesh contains a large number of slender triangles, and occasional defects. This mesh can be used as the starting point for a more complicated surface reconstruction algorithm which yields a very high detail and visually smooth surface, with good quality triangles, but far too many of them (Figure B.6(d)). Given such a highly detailed surface we found that defects are now easy to fix: the vertex density is high enough that a local resampling can be carried out around any defects without leading to large jagged artifacts. The final algorithm is a quadric edge decimation which aims to merge triangles whilst preserving the boundaries and overall topology of the mesh (Figure B.6(e)).

Figure B.7 gives some statistics about the meshes before and after this process: of primary interest are the minimum and mean quality statistics (which describe how close the triangles are to being equilateral: i.e. non-slender)

An alternative method is to use a meshing program specifically intended for this problem, called GAMER [162] (developed by the Holst group) which can take a PQR file and, when working correctly, produces high quality meshes. Unfortunately the program is prone to crashing on errors generated when the mesh is imperfect (without giving any option to ignore errors and produce output). We solved this by removing the assertions in the source code which detect mesh errors and thus forcing GAMER to produce output. Any errors (often due to a small number of bad vertices) can be fixed later using a simple procedure in Meshlab. So far we have always managed to create a good quality mesh using one or both of GAMER and Meshlab, but neither is 100% perfect. It may be noted that GAMER does not actually allow the user to set a solvent probe radius: the location of the surface is controlled by a hard-coded iso-parameterization constant in the GAMER source code, which controls the placement of the final surface layer via the marching cubes algorithm. Nonetheless, as distributed, GAMER appears to give very similar output to what would be expected from a solvent probe radius of about 1.5Å. Presumably this is not a coincidence since GAMER was written with biomolecular simulation in mind.

Again, some simple scripts are provided with BEEP to guide the user through the meshing process, but ultimate responsibility for the quality of the mesh lies with the user: poor quality meshes will lead to inaccurate results at best, and at worst BEEP may not be able to run if the mesh is extremely malformed (as various simple tests are carried out within the code to ensure that the mesh describes a consistent surface e.g. a mesh vertex must be connected to at least three triangles).

[3] or at least one which contains fewer defects than the original MSMS mesh, and on the occasions that defects remain they tend to be relatively minor (e.g. small holes)

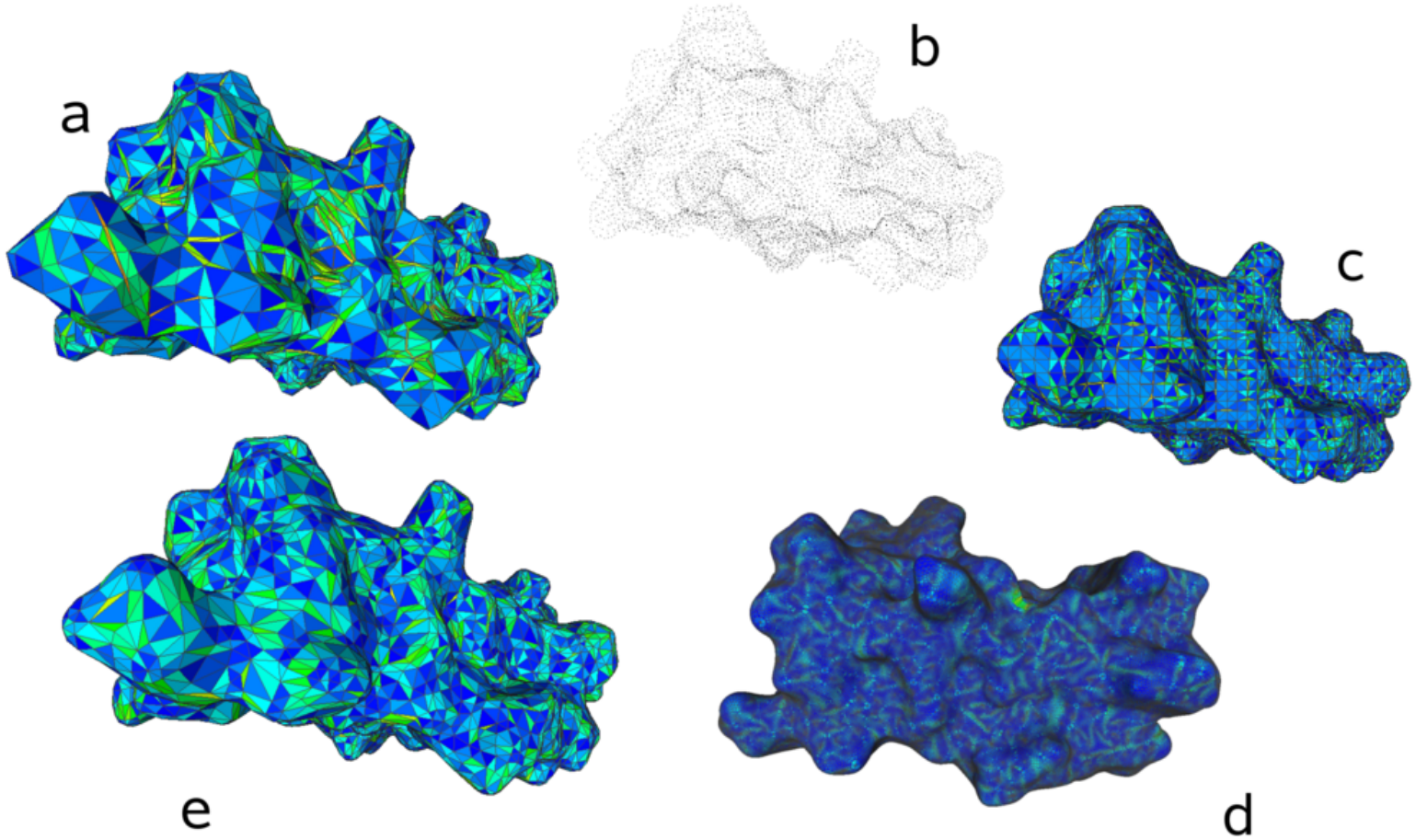


Figure B.6: Cleaning an MSMS mesh (a) for use with BEEP, using Meshlab, resulting in the much better quality mesh (e). The colours correspond to the triangle "quality" (ratio of area to maximum side length: blue is good, green is acceptable, orange/red is poor). A point cloud is (b) is generated from the MSMS mesh, which is then turned into a surface mesh using Poisson reconstruction (which in general seems to reliably produce an error-free mesh, but one with many low-quality slender triangles). The Poisson surface can be converted into a much smoother and higher quality representation (fewer slender triangles) (d) by a specialised refinement algorithm in Meshlab (repeated "butterfly subdivision"). Quadric edge decimation reduces the number of triangles (e) back to approximately the original number produced by MSMS.

	MSMS	high detail	final
Vertices:	5356	121893	5356
Edges:	16062	365673	16062
Faces:	10708	243782	10708
Topology			
Incompatible faces:	36	0	0
Duplicate faces:	0	0	0
Duplicate edges:	0	0	0
Boundary edges:	0	0	0
Non-manifold edges:	0	0	0
Edge per vertex avg.:	6	6	6
Edge per vertex max:	11	11	10
Faces per edge avg.:	2	2	2
Faces per edge max:	2	2	2
Geometry			
Volume:	7647	7666	7660
Area:	3340	3312	3322
BBox diagonal:	58.45	58.29	58.32
Face quality min:	0.02971	0.2001	0.3719
Face quality avg.:	0.7832	0.9657	0.9125
Face quality max:	1	1	1
Face area min:	2.45e-06	8.078e-05	0.0122
Face area avg.:	0.3119	0.01359	0.3102
Face area max:	1.506	0.1973	1.829
Edge length min:	0.0009994	0.003489	0.1211
Edge length avg.:	0.9512	0.1788	0.8993
Edge length max:	2.712	0.8228	2.837

Figure B.7: Comparison between the original MSMS mesh (Figure B.6(a)) and the final cleaned (Meshlab) mesh (Figure B.6(e)) (high detail intermediate (Figure B.6(d)) also shown): In the final mesh the topology has been fixed; all face quality metrics are improved with very little change to the volume and surface area of the entire mesh.

B.4 A Python interface to high-performance C++ code

In general BEEP is simple to use, and includes a Python interface for pre-processing of mesh files and to allow BEEP to be run from scripts. This combines the ease of use of Python (a high level object-oriented language) with the higher performance available with compiled C++ code. Most modern Linux distributions include Python (version 2.5 or greater) by default, so most users should have no difficulty with this requirement.

The program itself is intended to be easy to modify and extend using C++, with the various areas of complexity reasonably well modularised (e.g. a Mesh class to contain the various geometric properties of the mesh, a GMRES class which carries out the GMRES iterations, and a number of FMM classes which give the user easy access to FMM functionality without needing to worry too much about the implementation details).

B.5 Code acknowledgements & third-party libraries

The core of BEEP is written in standard C++ and should easily compile with modern compilers, such as the Intel C++ Compiler and the GNU compiler suite. For OpenMP support, version 10.1 or later of the Intel C++ Compiler is required, or version 4.2 (or later) of the GCC C++ compiler. BEEP was developed using GCC version 4.5.0.

BEEP was not developed in total isolation, and where possible used existing pieces of software to aid development. Specific pieces of code which have been incorporated into BEEP (and acknowledged within the code itself where appropriate) are:

- key parts of the FMM code were originally Fortran code developed by Jingfang Huang *et al.* [113] released under GNU Public License (GPL), then ported to C++ and modified by us (still technically a derivative work in terms of copyright and open-source software licensing). The porting of the FMM to C++ was carried out mostly in order to simplify the parallelisation process. Key FMM functions (for example those which calculated multipole expansions and those which convert to and from plane-wave representations) are directly translated from Fortran, however the program flow and octree data structures, within which the FMM takes place, are all entirely written from scratch by the author in the course of this project.
  (`http://fastmultipole.com`)

- the GMRES functions are based on code written by Christian Badura (based on the algorithm described by Saad [126]). The original code can be found either from the freely available lin_solver package or as part of the Cantera project (which is freely

available and released under the New BSD License).
(http://aam.mathematik.uni-freiburg.de/IAM/Research/projectskr/lin_solver/)
(http://code.google.com/p/cantera/)

- the BEM kernel function implementations are based on those by Benzhou Lu, see AFMPB (beta version, also released under GPL) [105]
(http://lsec.cc.ac.cn/~lubz/afmpb.html).

- Quadrature rules for triangular integration were obtained from John Burkardt from his personal web pages which contain an extensive collection of quadrature rules and numerical methods (code fragments released under GPL where applicable).
(http://people.sc.fsu.edu/~jburkardt/)

In addition BEEP has a number of dependencies on third-party software/libraries (all of which are open-source and freely available) namely:

- PDB2PQR *[157, 158]* (required for parameterizing PDB structures to PQR files)
(http://sourceforge.net/projects/pdb2pqr/)

- the C++ Boost project
(http://www.boost.org)

- the little template library (ltl)
(http://www.mpe.mpg.de/~drory/ltl/index.html)

- libarchive (v2.9) (which gives a C interface for compressed (gzip) tar archives)
(http://http://code.google.com/p/libarchive/)

- the Gnu Scientific Library (GSL)
(http://www.gnu.org/software/gsl/)

- Python (v2.5+) (for the Python interface)
(http://www.python.org)

The parallel versions of BEEP (described in the next chapter) require the following additional dependencies:

- for multi-CPU operation (e.g. on a cluster): Charm++ (available from the Parallel Computing Laboratory at University of Illinois)
http://charm.cs.uiuc.edu/

- for GPU accelerated code: a working OpenCL installation (e.g. install the CUDA toolkit from NVIDIA)
  `http://www.khronos.org/opencl/`

The BEEP package is open-source and released under the GNU Public License (GPL). Compiling BEEP from source is made straightforward by the use of GNU autotools: BEEP is distributed in a GNU-compliant package and compilation (under Linux) should be as simple as issuing "./configure && make" in the package directory. More advanced compilation options (e.g. for GPU-enabled versions) are detailed in the package documentation.

References

[1] W. F. van Gunsteren, J. Dolenc, and A. E. Mark, "Molecular simulation as an aid to experimentalists", *Current Opinion in Structural Biology*, vol. 18, no. 2, pp. 149–153, Apr. 2008, ISSN: 0959-440X.

[2] M. W. van der Kamp, K. E. Shaw, C. J. Woods, and A. J. Mulholland, "Biomolecular simulation and modelling: status, progress and prospects", *Journal of The Royal Society Interface*, vol. 5, no. Suppl 3, pp. 173–190, 2008.

[3] C. McInnes, "Virtual screening strategies in drug discovery", *Current Opinion in Chemical Biology*, vol. 11, no. 5, pp. 494–502, 2007, ISSN: 13675931.

[4] S. Huang and X. Zou, "Advances and challenges in Protein-Ligand docking", *International Journal of Molecular Sciences*, vol. 11, no. 8, pp. 3016–3034, 2010, ISSN: 1422-0067.

[5] D. Röthlisberger, O. Khersonsky, A. Wollacott, L. Jiang, J. DeChancie, J. Betker, J. Gallaher, E. Althoff, A. Zanghellini, O. Dym, et al., "Kemp elimination catalysts by computational enzyme design", *Nature*, vol. 453, no. 7192, pp. 190–195, 2008, ISSN: 0028-0836.

[6] J. E. Jones, "On the determination of molecular fields. II. from the equation of state of a gas", *Proceedings of the Royal Society of London. Series A*, vol. 106, no. 738, pp. 463–477, 1924.

[7] E. Arunan, G. Desiraju, R. Klein, J. Sadlej, S. Scheiner, I. Alkorta, D. Clary, R. Crabtree, J. Dannenberg, P. Hobza10, et al., "Definition of the hydrogen bond (iupac recommendations 2011)", *Pure Appl. Chem*, vol. 83, no. 8, pp. 1637–1641, 2011.

[8] Z. Hendsch and B. Tidor, "Do salt bridges stabilize proteins? a continuum electrostatic analysis", *Protein Science*, vol. 3, no. 2, pp. 211–226, 1994.

[9] D. Anderson, W. Becktel, and F. Dahlquist, "Ph-induced denaturation of proteins: a single salt bridge contributes 3-5 kcal/mol to the free energy of folding of t4 lysozyme", *Biochemistry*, vol. 29, no. 9, pp. 2403–2408, 1990.

[10] J. G. de la Torre, G. del Rio Echenique, and A. Ortega, "Improved calculation of rotational diffusion and intrinsic viscosity of bead models for macromolecules and nanoparticles.", *J Phys Chem B*, vol. 111, no. 5, pp. 955–961, 2007.

[11] J. G. de la Torre, M. L. Huertas, and B. Carrasco, "Calculation of hydrodynamic properties of globular proteins from their atomic-level structure.", *Biophys J*, vol. 78, no. 2, pp. 719–730, 2000.

[12] S. Aragon, "A precise boundary element method for macromolecular transport properties", *Journal of Computational Chemistry*, vol. 25, no. 9, pp. 1191–1205, 2004, ISSN: 1096-987X.

[13] S. Aragon and D. K. Hahn, "Precise boundary element computation of protein transport properties: diffusion tensors, specific volume, and hydration", *Biophysical Journal*, vol. 91, no. 5, pp. 1591–1603, Sep. 2006, ISSN: 0006-3495.

[14] J. Antosiewicz and J. A. McCammon, "Electrostatic and hydrodynamic orientational steering effects in enzyme-substrate association.", *Biophys J*, vol. 69, no. 1, pp. 57–65, 1995.

[15] J. Antosiewicz, J. M. Briggs, and J. A. McCammon, "Orientational steering in enzyme-substrate association: ionic strength dependence of hydrodynamic torque effects.", *Eur Biophys J*, vol. 24, no. 3, pp. 137–141, 1996.

[16] D. Brune and S. Kim, "Hydrodynamic steering effects in protein association.", *Proc Natl Acad Sci U S A*, vol. 91, no. 8, pp. 2930–2934, 1994.

[17] D. S. Goodsell, "Inside a living cell", *Trends in Biochemical Sciences*, vol. 16, no. 6, pp. 203–206, Jun. 1991, PMID: 1891800, ISSN: 0968-0004.

[18] R. J. Ellis, "Macromolecular crowding: obvious but underappreciated", *Trends in Biochemical Sciences*, vol. 26, no. 10, pp. 597–604, Oct. 2001, ISSN: 0968-0004.

[19] A. H. Elcock, "Models of macromolecular crowding effects and the need for quantitative comparisons with experiment", *Current Opinion in Structural Biology*, vol. 20, no. 2, pp. 196–206, Apr. 2010, PMID: 20167475, ISSN: 1879-033X.

[20] D. Bray, "Protein molecules as computational elements in living cells", *Nature*, vol. 376, no. 6538, pp. 307–312, 1995, ISSN: 0028-0836.

[21] L. M. Loew and J. C. Schaff, "The virtual cell: a software environment for computational cell biology", *Trends in Biotechnology*, vol. 19, no. 10, pp. 401–406, Oct. 2001, ISSN: 0167-7799.

[22] R. Randhawa, C. A. Shaffer, and J. J. Tyson, "Model aggregation: a building-block approach to creating large macromolecular regulatory networks", *Bioinformatics*, vol. 25, no. 24, pp. 3289 –3295, Dec. 2009.

[23] J. S. van Zon and P. R. ten Wolde, "Green's-function reaction dynamics: a particle-based approach for simulating biochemical networks in time and space", *The Journal of Chemical Physics*, vol. 123, no. 23, pp. 234 910–16, Dec. 2005.

[24] A. L. Tournier, P. W. Fitzjohn, and P. A. Bates, "Probability-based model of protein-protein interactions on biological timescales.", *Algorithms Mol Biol*, vol. 1, p. 25, 2006.

[25] J. Fenner, B Brook, G Clapworthy, P. Coveney, V Feipel, H Gregersen, D. Hose, P Kohl, P Lawford, K. McCormack, D Pinney, S. Thomas, S. V. S. Jan, S Waters, and M Viceconti, "The EuroPhysiome, STEP and a roadmap for the virtual physiological human", *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 366, no. 1878, pp. 2979 –2999, 2008.

[26] C. M. Lloyd, M. D. B. Halstead, and P. F. Nielsen, "CellML: its future, present and past", *Progress in Biophysics and Molecular Biology*, vol. 85, no. 2-3, pp. 433–450, 2004, ISSN: 0079-6107.

[27] D. A. Leach, *Molecular Modelling: Principles and Applications*, 2nd ed. Prentice Hall, 2001, ISBN: 0582382106.

[28] A. Warshel and M. Levitt, "Theoretical studies of enzymic reactions: dielectric, electrostatic and steric stabilization of the carbonium ion in the reaction of lysozyme", *Journal of Molecular Biology*, vol. 103, no. 2, pp. 227–249, May 1976, ISSN: 0022-2836.

[29] S. A. Adcock and J. A. McCammon, "Molecular dynamics: survey of methods for simulating the activity of proteins.", *Chem Rev*, vol. 106, no. 5, pp. 1589–1615, 2006.

[30] D. Case, T. Cheatham III, T. Darden, H. Gohlke, R. Luo, K. Merz Jr, A. Onufriev, C. Simmerling, B. Wang, and R. Woods, "The Amber biomolecular simulation programs", *Journal of computational chemistry*, vol. 26, no. 16, pp. 1668–1688, 2005, ISSN: 1096-987X.

[31] J. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. Skeel, L. Kale, and K. Schulten, "Scalable molecular dynamics with NAMD", *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005, ISSN: 1096-987X.

[32] E. Lindahl, B. Hess, and D. van der Spoel, "GROMACS 3.0: a package for molecular simulation and trajectory analysis", *Journal of Molecular Modeling*, vol. 7, no. 8, pp. 306–317, 2001, ISSN: 1610-2940.

[33] B. Brooks, R. Bruccoleri, B. Olafson, D. States, S. Swaminathan, and M. Karplus, "CHARMM - A program for macromolecular energy, minimization, and dynamics calculations", English, *Journal of Computational Chemistry*, vol. 4, no. 2, 187–217, 1983, ISSN: 0192-8651.

[34] J. W. Ponder and F. M. Richards, "An efficient newton-like method for molecular mechanics energy minimization of large molecules", *Journal of Computational Chemistry*, vol. 8, no. 7, pp. 1016–1024, 1987.

[35] A. D. MacKerell, "Empirical force fields for biological macromolecules: overview and issues", *Journal of Computational Chemistry*, vol. 25, no. 13, pp. 1584–1604, 2004, ISSN: 1096-987X.

[36] S. A. Showalter and R. Brüschweiler, "Validation of molecular dynamics simulations of biomolecules using NMR spin relaxation as benchmarks: application to the AMBER99SB force field", *Journal of Chemical Theory and Computation*, vol. 3, no. 3, pp. 961–975, May 2007.

[37] E. J. Thompson, A. J. DePaul, S. S. Patel, and E. J. Sorin, "Evaluating molecular mechanical potentials for helical peptides and proteins", *PLoS ONE*, vol. 5, no. 4, J. Langowski, Ed., e10056, 2010, ISSN: 1932-6203.

[38] B. Kim, T. Young, E. Harder, R. A. Friesner, and B. J. Berne, "Structure and dynamics of the solvation of bovine pancreatic trypsin inhibitor in explicit water: a comparative study of the effects of solvent and protein polarizability", *The Journal of Physical Chemistry. B*, vol. 109, no. 34, pp. 16 529–16 538, Sep. 2005, PMID: 16853101, ISSN: 1520-6106.

[39] A. T. Brunger and P. D. Adams, "Molecular dynamics applied to x-ray structure refinement", *Accounts of Chemical Research*, vol. 35, no. 6, pp. 404–412, 2002.

[40] A. Fersht and V. Daggett, "Protein folding and unfolding at atomic resolution", *Cell*, vol. 108, no. 4, pp. 573–582, 2002.

[41] E. Lindahl and M. Sansom, "Membrane proteins: molecular dynamics simulations", *Current opinion in structural biology*, vol. 18, no. 4, pp. 425–431, 2008.

[42] J. A. McCammon, B. R. Gelin, and M. Karplus, "Dynamics of folded proteins", *Nature*, vol. 267, no. 5612, pp. 585–590, Jun. 1977.

[43] L. Monticelli, S. K. Kandasamy, X. Periole, R. G. Larson, D. P. Tieleman, and S. Marrink, "The MARTINI Coarse-Grained force field: extension to proteins", *Journal of Chemical Theory and Computation*, vol. 4, no. 5, pp. 819–834, May 2008, ISSN: 1549-9618.

[44] V. A. Voelz, G. R. Bowman, K. Beauchamp, and V. S. Pande, "Molecular simulation of ab initio protein folding for a millisecond folder NTL9(1â39)", *Journal of the American Chemical Society*, vol. 132, no. 5, pp. 1526–1528, Feb. 2010.

[45] P. Maragakis, K. Lindorff-Larsen, M. Eastwood, R. Dror, J. Klepeis, I. Arkin, M. Jensen, H. Xu, N. Trbovic, R. Friesner, et al., "Microsecond molecular dynamics simulation shows effect of slow loop dynamics on backbone amide order parameters of proteins", *The Journal of Physical Chemistry B*, vol. 112, no. 19, pp. 6155–6158, 2008.

[46] J. A. Izaguirre, D. P. Catarello, J. M. Wozniak, and R. D. Skeel, "Langevin stabilization of molecular dynamics", *The Journal of Chemical Physics*, vol. 114, no. 5, p. 2090, 2001, ISSN: 00219606.

[47] S. Liu and M. Muthukumar, "Langevin dynamics simulation of counterion distribution around isolated flexible polyelectrolyte chains", *The Journal of Chemical Physics*, vol. 116, no. 22, p. 9975, 2002, ISSN: 00219606.

[48] D. L. Ermak and J. A. McCammon, "Brownian dynamics with hydrodynamic interactions", *The Journal of Chemical Physics*, vol. 69, no. 4, p. 1352, 1978, ISSN: 00219606.

[49] P. J. Rossky, J. D. Doll, and H. L. Friedman, "Brownian dynamics as smart monte carlo simulation", *The Journal of Chemical Physics*, vol. 69, no. 10, pp. 4628–4633, 1978.

[50] R. C. Wade, "Simulation of enzyme-substrate interactions: the diffusional encounter step.", *Acta Biochim Pol*, vol. 42, no. 4, pp. 419–425, 1995.

[51] G. A. Huber and S. Kim, "Weighted-ensemble brownian dynamics simulations for protein association reactions.", *Biophys J*, vol. 70, no. 1, pp. 97–110, 1996.

[52] R. Gabdoulline and R. Wade, "Simulation of the diffusional association of barnase and barstar", *Biophysical Journal*, vol. 72, no. 5, pp. 1917–1929, May 1997, ISSN: 0006-3495.

[53] R. R. Gabdoulline and R. C. Wade, "Brownian dynamics simulation of protein-protein diffusional encounter.", *Methods*, vol. 14, no. 3, pp. 329–341, 1998.

[54] A. H. Elcock, R. R. Gabdoulline, R. C. Wade, and J. McCammon, "Computer simulation of protein-protein association kinetics: acetylcholinesterase-fasciculin", *Journal of Molecular Biology*, vol. 291, no. 1, pp. 149–162, Sep. 1999, ISSN: 0022-2836.

[55] A. Rojnuckarin, D. R. Livesay, and S. Subramaniam, "Bimolecular reaction simulation using weighted ensemble brownian dynamics and the university of houston brownian dynamics program.", *Biophys J*, vol. 79, no. 2, pp. 686–693, 2000.

[56] R. R. Gabdoulline and R. C. Wade, "Protein-protein association: investigation of factors influencing association rates by brownian dynamics simulations.", *J Mol Biol*, vol. 306, no. 5, pp. 1139–1155, 2001.

[57] M. X. Fernandes and J. G. de la Torre, "Brownian dynamics simulation of rigid particles of arbitrary shape in external fields.", *Biophys J*, vol. 83, no. 6, pp. 3039–3048, 2002.

[58] R. R. Gabdoulline and R. C. Wade, "Biomolecular diffusional association", *Current Opinion in Structural Biology*, vol. 12, no. 2, pp. 204–213, Apr. 2002, ISSN: 0959-440X.

[59] M. Schlosshauer and D. Baker, "Realistic protein-protein association rates from a simple diffusional model neglecting long-range interactions, free energy barriers, and landscape ruggedness.", *Protein Sci*, vol. 13, no. 6, pp. 1660–1669, 2004.

[60] S. R. McGuffee and A. H. Elcock, "Atomically detailed simulations of concentrated protein solutions: the effects of salt, pH, point mutations, and protein concentration in simulations of 1000-Molecule systems", *Journal of the American Chemical Society*, vol. 128, no. 37, pp. 12 098–12 110, 2006.

[61] S. McGuffee and A. Elcock, "Diffusion, crowding & protein stability in a dynamic molecular model of the bacterial cytoplasm", *PLoS Comput Biol*, vol. 6, no. 3, e1000694, Mar. 2010.

[62] J. Madura, J. Briggs, R. Wade, M. Davis, B. Luty, A. Ilin, J. Antosiewicz, M. Gilson, B. Bagheri, L. Scott, et al., "Electrostatics and diffusion of molecules in solution: simulations with the University of Houston Brownian Dynamics program", *Computer Physics Communications*, vol. 91, no. 1-3, pp. 57–95, 1995.

[63] P. Debye and E. Hückel, "Zur theorie der elektrolyte", *Phys. Z.*, vol. 24, p. 185, 1923.

[64] M. E. Davis and J. A. McCammon, "Electrostatics in biomolecular structure and dynamics", *Chemical Reviews*, vol. 90, no. 3, pp. 509–521, May 1990.

[65] A. Warshel, P. K. Sharma, M. Kato, and W. W. Parson, "Modeling electrostatic effects in proteins", *Biochimica et Biophysica Acta (BBA) - Proteins & Proteomics*, vol. 1764, no. 11, pp. 1647–1676, Nov. 2006, ISSN: 1570-9639.

[66] J. W. Ponder, C. Wu, P. Ren, V. S. Pande, J. D. Chodera, M. J. Schnieders, I. Haque, D. L. Mobley, D. S. Lambrecht, R. A. DiStasio, M. Head-Gordon, G. N. I. Clark, M. E. Johnson, and T. Head-Gordon, "Current status of the AMOEBA polarizable force field", *The Journal of Physical Chemistry B*, vol. 114, no. 8, pp. 2549–2564, Mar. 2010, ISSN: 1520-6106.

[67] S. O. Yesylevskyy, L. V. Schäfer, D. Sengupta, and S. J. Marrink, "Polarizable water model for the coarse-grained martini force field", *PLoS Comput Biol*, vol. 6, no. 6, e1000810, Jun. 2010.

[68] J. Warwicker and H. C. Watson, "Calculation of the electric potential in the active site cleft due to [alpha]-helix dipoles", *Journal of Molecular Biology*, vol. 157, no. 4, pp. 671–679, Jun. 1982, ISSN: 0022-2836.

[69] J. Warwicker, "Continuum dielectric modelling of the protein-solvent system, and calculation of the long-range electrostatic field of the enzyme phosphoglycerate mutase", *Journal of Theoretical Biology*, vol. 121, no. 2, pp. 199–210, Jul. 1986, ISSN: 0022-5193.

[70] M. K. Gilson, A. Rashin, R. Fine, and B. Honig, "On the calculation of electrostatic interactions in proteins", *Journal of Molecular Biology*, vol. 184, no. 3, pp. 503–516, Aug. 1985, ISSN: 0022-2836.

[71] M. K. Gilson and B. H. Honig, "Calculation of electrostatic potentials in an enzyme active site", *Nature*, vol. 330, no. 6143, pp. 84–86, Nov. 1987.

[72] M. K. Gilson, K. A. Sharp, and B. H. Honig, "Calculating the electrostatic potential of molecules in solution: method and error assessment", *Journal of Computational Chemistry*, vol. 9, no. 4, pp. 327–335, 1988, ISSN: 1096-987X.

[73] K. A. Sharp and B. Honig, "Calculating total electrostatic energies with the nonlinear Poisson-Boltzmann equation", *The Journal of Physical Chemistry*, vol. 94, no. 19, pp. 7684–7692, 1990.

[74] D. Bashford and D. A. Case, "Generalized Born models of macromolecular solvation effects", *Annual Review of Physical Chemistry*, vol. 51, no. 1, pp. 129–152, 2000.

[75] J. A. Wagoner and N. A. Baker, "Assessing implicit models for nonpolar mean solvation forces: the importance of dispersion and volume terms.", *Proc Natl Acad Sci U S A*, vol. 103, no. 22, pp. 8331–8336, 2006.

[76] A. Nicholls, D. L. Mobley, J. P. Guthrie, J. D. Chodera, C. I. Bayly, M. D. Cooper, and V. S. Pande, "Predicting Small-Molecule solvation free energies: an informal blind test for computational chemistry", *Journal of Medicinal Chemistry*, vol. 51, no. 4, pp. 769–779, Feb. 2008.

[77] C. Tan, L. Yang, and R. Luo, "How well does Poisson-Boltzmann implicit solvent agree with explicit solvent? a quantitative analysis", *The Journal of Physical Chemistry. B*, vol. 110, no. 37, pp. 18 680–18 687, Sep. 2006, PMID: 16970499, ISSN: 1520-6106.

[78] M. Born, "Volumen und hydratationswärme der ionen", *Zeitschrift für Physik A Hadrons and Nuclei*, vol. 1, no. 1, pp. 45–48, 1920, ISSN: 0939-7922.

[79] W. Still, A. Tempczyk, R. Hawley, and T. Hendrickson, "Semianalytical treatment of solvation for molecular mechanics and dynamics", *Journal of the American Chemical Society*, vol. 112, no. 16, pp. 6127–6129, 1990, ISSN: 0002-7863.

[80] M. Feig and C. L. Brooks, "Recent advances in the development and application of implicit solvent models in biomolecule simulations", *Current Opinion in Structural Biology*, vol. 14, no. 2, pp. 217–224, Apr. 2004, ISSN: 0959-440X.

[81] H. Tjong and H. Zhou, "GBr6: a parameterization-free, accurate, analytical generalized Born method", *The Journal of Physical Chemistry B*, vol. 111, no. 11, pp. 3055–3061, 2007.

[82] H. Tjong and H. X. Zhou, "Accurate calculations of binding, folding, and transfer free energies by a scaled Generalized Born method", *Journal of Chemical Theory and Computation*, vol. 4, no. 10, pp. 1733–1744, 2008.

[83] M. Feig, A. Onufriev, M. S. Lee, W. Im, D. A. Case, and C. L. Brooks, "Performance comparison of generalized Born and Poisson methods in the calculation of electrostatic solvation energies for protein structures", *Journal of Computational Chemistry*, vol. 25, no. 2, pp. 265–284, 2004, ISSN: 1096-987X.

[84] W. Zhao, G. Xu, and C. Bajaj, "An algebraic spline model of molecular surfaces", in *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, ser. SPM '07, Beijing, China: ACM, 2007, pp. 297–302, ISBN: 978-1-59593-666-0. DOI: http://doi.acm.org/10.1145/1236246.1236288. [Online]. Available: http://doi.acm.org/10.1145/1236246.1236288.

[85] A. Warshel and S. T. Russell, "Calculations of electrostatic interactions in biological systems and in solutions", *Quarterly Reviews of Biophysics*, vol. 17, no. 03, pp. 283–422, 1984.

[86] C. N. Schutz and A. Warshel, "What are the dielectric "constants" of proteins and how to validate electrostatic models?", *Proteins: Structure, Function, and Bioinformatics*, vol. 44, no. 4, pp. 400–417, 2001, ISSN: 1097-0134.

[87] M. E. Davis, "The inducible multipole solvation model: a new model for solvation effects on solute electrostatics", *The Journal of Chemical Physics*, vol. 100, no. 7, p. 5149, 1994, ISSN: 00219606.

[88] R. R. Gabdoulline and R. C. Wade, "Effective charges for macromolecules in solvent", *The Journal of Physical Chemistry*, vol. 100, no. 9, pp. 3868–3878, Jan. 1996.

[89] R. R. Gabdoulline and R. C. Wade, "On the contributions of diffusion and thermal activation to electron transfer between phormidium laminosum plastocyanin and cytochrome f: brownian dynamics simulations with explicit modeling of nonpolar desolvation interactions and electron transfer events", *Journal of the American Chemical Society*, vol. 131, no. 26, pp. 9230–9238, Jul. 2009.

[90] N. A. Baker, D. Sept, S. Joseph, M. J. Holst, and J. A. McCammon, "Electrostatics of nanosystems: application to microtubules and the ribosome", *Proceedings of the National Academy of Sciences of the United States of America*, vol. 98, no. 18, pp. 10037 –10 041, 2001.

[91] M. E. Davis and J. A. McCammon, "Calculating electrostatic forces from grid-calculated potentials", *Journal of Computational Chemistry*, vol. 11, no. 3, pp. 401–409, 1990, ISSN: 1096-987X.

[92] D. Sitkoff, K. Sharp, and B. Honig, "Accurate calculation of hydration free energies using macroscopic solvent models", *The Journal of Physical Chemistry*, vol. 98, no. 7, pp. 1978–1988, Feb. 1994.

[93] W. Im, D. Beglov, and B. Roux, "Continuum solvation model: computation of electrostatic forces from numerical solutions to the Poisson-Boltzmann equation", *Computer Physics Communications*, vol. 111, no. 1-3, pp. 59–75, Jun. 1998, ISSN: 0010-4655.

[94] R. J. Zauhar and R. S. Morgan, "A new method for computing the macromolecular electric potential", *Journal of Molecular Biology*, vol. 186, no. 4, pp. 815–820, Dec. 1985, ISSN: 0022-2836.

[95] R. J. Zauhar and A. Varnek, "A fast and space-efficient boundary element method for computing electrostatic and hydration effects in large molecules", *Journal of Computational Chemistry*, vol. 17, no. 7, pp. 864–877, 1996, ISSN: 1096-987X.

[96] A. Boschitsch, M. Fenley, and W. Olson, "A fast adaptive multipole algorithm for calculating screened Coulomb (Yukawa) interactions", *Journal of Computational Physics*, vol. 151, no. 1, pp. 212–241, 1999.

[97] A. H. Boschitsch, M. O. Fenley, and H. Zhou, "Fast boundary element method for the linear Poisson-Boltzmann equation", *The Journal of Physical Chemistry B*, vol. 106, no. 10, pp. 2741–2754, Mar. 2002.

[98] A. J. Bordner and G. A. Huber, "Boundary element solution of the linear Poisson-Boltzmann equation and a multipole method for the rapid calculation of forces on macromolecules in solution", *Journal of Computational Chemistry*, vol. 24, no. 3, pp. 353–367, Feb. 2003, PMID: 12548727, ISSN: 0192-8651.

[99] B. Lu, D. Zhang, and J. A. McCammon, "Computation of electrostatic forces between solvated molecules determined by the poisson–boltzmann equation using a boundary element method", *The Journal of Chemical Physics*, vol. 122, no. 21, 214102, p. 214 102, 2005.

[100] B. Lu, X. Cheng, T. Hou, and J. A. McCammon, "Calculation of the Maxwell stress tensor and the Poisson-Boltzmann force on a solvated molecular surface using hypersingular boundary integrals.", *J Chem Phys*, vol. 123, no. 8, p. 084 904, 2005.

[101] B. Lu, X. Cheng, J. Huang, and J. A. McCammon, "Order N algorithm for computation of electrostatic interactions in biomolecular systems.", *Proc Natl Acad Sci U S A*, vol. 103, no. 51, pp. 19 314–19 319, 2006.

[102] B. Lu and J. A. McCammon, "Improved boundary element methods for Poisson-Boltzmann electrostatic potential and force calculations", *Journal of Chemical Theory and Computation*, vol. 3, no. 3, pp. 1134–1142, May 2007.

[103] B. Lu, X. Cheng, and J. A. McCammon, ""New-version-fast-multipole-method" accelerated electrostatic calculations in biomolecular systems", *Journal of Computational Physics*, vol. 226, pp. 1348–1366, Oct. 2007, ACM ID: 1290427, ISSN: 0021-9991.

[104] B. Z. Lu, Y. C. Zhou, M. J. Holst, and J. A. McCammon, "Recent progress in numerical methods for the Poisson-Boltzmann equation in biophysical applications", *Communications In Computational Physics*, vol. 3, no. 5, pp. 973–1009, May 2008.

[105] B. Lu, X. Cheng, J. Huang, and J. A. McCammon, "AFMPB: an adaptive fast multipole Poisson-Boltzmann solver for calculating electrostatics in biomolecular systems", *Computer Physics Communications*, vol. 181, no. 6, pp. 1150–1160, 2010, ISSN: 00104655.

[106] M. D. Altman, J. P. Bardhan, J. K. White, and B. Tidor, "Accurate solution of multi-region continuum biomolecule electrostatic problems using the linearized Poisson-Boltzmann equation with curved boundary elements", *Journal of Computational Chemistry*, vol. 30, no. 1, pp. 132–153, Jan. 2009, PMID: 18567005, ISSN: 1096-987X.

[107] J. P. Bardhan, M. D. Altman, D. J. Willis, S. M. Lippow, B. Tidor, and J. K. White, "Numerical integration techniques for curved-element discretizations of molecule-solvent interfaces.", *J Chem Phys*, vol. 127, no. 1, p. 014 701, 2007.

[108] J. P. Bardhan, R. S. Eisenberg, and D. Gillespie, "Discretization of the induced-charge boundary integral equation", *Physical Review E*, vol. 80, no. 1, p. 011 906, Jul. 2009.

[109] J. P. Bardhan, "Numerical solution of boundary-integral equations for molecular electrostatics", *The Journal of Chemical Physics*, vol. 130, no. 9, p. 094102, 2009, ISSN: 00219606.

[110] R. Yokota, J. P. Bardhan, M. G. Knepley, L. A Barba, and T. Hamada, "Biomolecular electrostatics simulation with a parallel FMM-based BEM, using up to 512 GPUs", *Arxiv preprint arXiv:1007.4591*, Jul. 2010.

[111] R. Bharadwaj, A. Windemuth, S. Sridharan, B. Honig, and A. Nicholls, "The fast multipole boundary element method for molecular electrostatics: an optimal approach for large systems", *Journal of Computational Chemistry*, vol. 16, no. 7, pp. 898–913, 1995, ISSN: 1096-987X.

[112] L. F. Greengard and J. Huang, "A new version of the fast multipole method for screened coulomb interactions in three dimensions", *Journal of Computational Physics*, vol. 180, no. 2, pp. 642–658, Aug. 2002, ISSN: 0021-9991.

[113] J. Huang, J. Jia, and B. Zhang, "FMM-Yukawa: an adaptive fast multipole method for screened coulomb interactions", *Computer Physics Communications*, vol. 180, no. 11, pp. 2331–2338, Nov. 2009, ISSN: 0010-4655.

[114] A. J. Juffer, E. F. F. Botta, B. A. M. van Keulen, A. van der Ploeg, and H. J. C. Berendsen, "The electric potential of a macromolecule in a solvent: a fundamental approach", *Journal of Computational Physics*, vol. 97, pp. 144–171, Nov. 1991, ACM ID: 131706, ISSN: 0021-9991.

[115] S. Kuo, B. Tidor, and J. White, "A meshless, spectrally accurate, integral equation solver for molecular surface electrostatics", *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 4, 6:1–6:30, Apr. 2008, ACM ID: 1350766, ISSN: 1550-4832.

[116] J. Tausch, J. Wang, and J. White, "Improved integral formulations for fast 3-D method-of-moments solvers", *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 20, no. 12, pp. 1398–1405, 2001, ISSN: 0278-0070.

[117] J. M. J. Swanson, S. A. Adcock, and J. A. McCammon, "Optimized radii for Poisson-Boltzmann calculations with the AMBER force field", *Journal of Chemical Theory and Computation*, vol. 1, no. 3, pp. 484–493, May 2005.

[118] J. M. J. Swanson, J. A. Wagoner, N. A. Baker, and J. A. McCammon, "Optimizing the Poisson dielectric boundary with explicit solvent forces and energies: lessons learned with atom-centered dielectric functions", *Journal of Chemical Theory and Computation*, vol. 3, no. 1, pp. 170–183, Jan. 2007.

[119] M. Nina, D. Beglov, and B. Roux, "Atomic radii for continuum electrostatics calculations based on molecular dynamics free energy simulations", *The Journal of Physical Chemistry B*, vol. 101, no. 26, pp. 5239–5248, Jun. 1997.

[120] M. Nina, W. Im, and B. Roux, "Optimized atomic radii for protein continuum electrostatics solvation forces", *Biophysical Chemistry*, vol. 78, no. 1-2, pp. 89–96, Apr. 1999, ISSN: 0301-4622.

[121] S. Yu, W. Geng, and G. W. Wei, "Treatment of geometric singularities in implicit solvent models", *The Journal of Chemical Physics*, vol. 126, no. 24, p. 244 108, Jun. 2007, PMID: 17614538, ISSN: 0021-9606.

[122] W. Geng, S. Yu, and G. Wei, "Treatment of charge singularities in implicit solvent models", *The Journal of Chemical Physics*, vol. 127, no. 11, p. 114 106, 2007, ISSN: 00219606.

[123] J. D. Jackson, *Classical electrodynamics (3rd Edition)*. John Wiley & Sons, Inc., 1999, p. 808, ISBN: 0-471-30932-X.

[124] J. A. Stratton, *Electromagnetic theory*. McGraw-Hill Book Company Inc., 1941, ISBN: 0070621500.

[125] I. Stakgold, *Boundary value problems of mathematical physics (vol. 2)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000, ISBN: 0-89871-456-7.

[126] Y. Saad and M. Schultz, "Gmres: a generalized minimal residual algorithm for solving nonsymmetric linear systems.", *SIAM J. Sci. Stat. Comput.*, vol. 7, no. 3, pp. 856–869, 1986.

[127] L. Greengard, J. Huang, V. Rokhlin, and S. Wandzura, "Accelerating fast multipole methods for the helmholtz equation at low frequencies", *Computational Science & Engineering, IEEE*, vol. 5, no. 3, pp. 32–38, 1998, ISSN: 1070-9924.

[128] L. Greengard and V. Rokhlin, "A new version of the fast multipole method for the laplace equation in three dimensions", *Acta numerica*, vol. 6, pp. 229–269, 1997.

[129] M. Abramowitz and I. Stegun, *Handbook of mathematical functions with formulas, graphs, and mathematical tables*. Dover publications, 1964, ISBN: 0486612724.

[130] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, F. Rossi, and R. Price, "GNU scientific library reference manual: (v. 1.12)", *Bristol, UK: Network Theory Ltd*, 2009.

[131] L. Biedenharn, J. Louck, and P. Carruthers, *Angular Momentum in Quantum Physics: Theory and Applications (Encyclopedia of Mathematics and its Applications 8)*. Reading, MA: Addison-Wesley, 1981.

[132] H. Cheng, V. Rokhlin, and N. Yarvin, "Nonlinear optimization, quadrature, and interpolation", *SIAM Journal on Optimization*, vol. 9, p. 901, 1999.

[133] D. Dunavant, "High degree efficient symmetrical gaussian quadrature rules for the triangle", *International journal for numerical methods in engineering*, vol. 21, no. 6, pp. 1129–1148, 1985.

[134] J. Lyness and D. Jespersen, "Moderate degree symmetric quadrature rules for the triangle", *IMA Journal of Applied Mathematics*, vol. 15, no. 1, p. 19, 1975.

[135] M. Guiggiani, G. Krishnasamy, T. Rudolphi, and F. Rizzo, "A general algorithm for the numerical solution of hypersingular boundary integral equations", *ASME J. Appl. Mech*, vol. 59, no. 3, pp. 604–614, 1992.

[136] M. K. Gilson, M. E. Davis, B. A. Luty, and J. A. McCammon, "Computation of electrostatic forces on solvated molecules using the Poisson-Boltzmann equation", *The Journal of Physical Chemistry*, vol. 97, no. 14, pp. 3591–3600, Apr. 1993.

[137] J. Che, J. Dzubiella, B. Li, and J. McCammon, "Electrostatic free energy and its variations in implicit solvent models", *The Journal of Physical Chemistry B*, vol. 112, no. 10, pp. 3058–3069, 2008.

[138] V. Ferraro, *Electromagnetic theory*. Athlone Press, 1954, ISBN: 1446514110.

[139] J. Kirkwood, "Theory of solutions of molecules containing widely separated charges with special application to zwitterions", *The Journal of Chemical Physics*, vol. 2, p. 351, 1934.

[140] T. Hill, "Influence of electrolyte on effective dielectric constants in enzymes, proteins and other molecules", *The Journal of Physical Chemistry*, vol. 60, no. 2, pp. 253–255, 1956.

[141] J. D. Sloan, *High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI*, 1st edition. O'Reilly Media, 2004, ch. 1, pp. 14–15, ISBN: 0596005709.

[142] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming", *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

[143] L. Kale and S. Krishnan, "CHARM++: a portable concurrent object oriented system based on C++", in *ACM Sigplan Notices*, ACM, vol. 28, 1993, pp. 91–108.

[144] *Charm++ (University of Illinois at Urbana-Champaign): A language/paradigm for parallel objects*, `http://charm.cs.uiuc.edu/research/charm/`.

[145] *Valgrind: an instrumentation framework for building dynamic analysis tools.* `http://valgrind.org/`.

[146] *CUDA: NVIDIA's parallel computing architecture*, `http://www.nvidia.com/object/cuda_home_new.html`.

[147] *ATI Stream: GPU and CPU Technology for Accelerated Computing*, `http://www.amd.com/us/products/technologies/stream-technology/pages/stream-technology.aspx`.

[148] *OpenCL - The open standard for parallel programming of heterogeneous systems*, `http://www.khronos.org/opencl/`.

[149] C. Fünfzig, K. Müller, D. Hansford, and G. Farin, "Png1 triangles for tangent plane continuous surfaces on the gpu", in *Proceedings of graphics interface 2008*, Canadian Information Processing Society, 2008, pp. 219–226.

[150] R. Tan, T. Truong, J. McCammon, and J. Sussman, "Acetylcholinesterase: electrostatic steering increases the rate of ligand binding", *Biochemistry*, vol. 32, no. 2, pp. 401–403, 1993.

[151] Z. Radić, P. Kirchhoff, D. Quinn, J. McCammon, and P. Taylor, "Electrostatic influence on the kinetics of ligand binding to acetylcholinesterase", *Journal of Biological Chemistry*, vol. 272, no. 37, p. 23 265, 1997.

[152] T. Shen, K. Tai, R. Henchman, and J. McCammon, "Molecular dynamics of acetylcholinesterase", *Accounts of chemical research*, vol. 35, no. 6, pp. 332–340, 2002.

[153] J. Bui, Z. Radic, P. Taylor, and J. McCammon, "Conformational transitions in protein-protein association: binding of fasciculin-2 to acetylcholinesterase", *Biophysical journal*, vol. 90, no. 9, pp. 3280–3287, 2006.

[154] H. Zhou, "Interactions of macromolecules with salt ions: an electrostatic theory for the Hofmeister effect", *Proteins: Structure, Function, and Bioinformatics*, vol. 61, no. 1, pp. 69–78, 2005.

[155] A. Thomas and A. Elcock, "Direct observation of salt effects on molecular interactions through explicit-solvent molecular dynamics simulations: differential effects on electrostatic and hydrophobic interactions and comparisons to Poisson-Boltzmann theory", *Journal of the American Chemical Society*, vol. 128, no. 24, pp. 7796–7806, 2006.

[156] M. Konopka, I. Shkel, S. Cayley, M. Record, and J. Weisshaar, "Crowding and confinement effects on protein diffusion in vivo", *Journal of bacteriology*, vol. 188, no. 17, p. 6115, 2006.

[157] T. J. Dolinsky, J. E. Nielsen, J. A. McCammon, and N. A. Baker, "PDB2PQR: an automated pipeline for the setup of Poisson-Boltzmann electrostatics calculations", *Nucleic Acids Research*, vol. 32, no. Web Server, W665–W667, 2004, ISSN: 0305-1048.

[158] T. J. Dolinsky, P. Czodrowski, H. Li, J. E. Nielsen, J. H. Jensen, G. Klebe, and N. A. Baker, "PDB2PQR: expanding and upgrading automated preparation of biomolecular structures for molecular simulations", *Nucleic Acids Research*, vol. 35, no. Web Server, W522–W525, 2007, ISSN: 0305-1048.

[159] D. Xu, C. Tsai, and R. Nussinov, "Hydrogen bonds and salt bridges across protein-protein interfaces.", *Protein engineering*, vol. 10, no. 9, p. 999, 1997.

[160] R. Salari and L. Chong, "Desolvation costs of salt bridges across protein binding interfaces: similarities and differences between implicit and explicit solvent models", *The Journal of Physical Chemistry Letters*, vol. 1, pp. 2844–2848, 2010.

[161] H. Li, A. Robertson, and J. Jensen, "Very fast empirical prediction and rationalization of protein pka values", *Proteins-Structure Function and Bioinformatics*, vol. 61, no. 4, pp. 704–721, 2005.

[162] Z. Yu, M. J. Holst, Y. Cheng, and J. A. McCammon, "Feature-preserving adaptive mesh generation for molecular shape modeling and simulation", *Journal of Molecular*

Graphics & Modelling, vol. 26, no. 8, pp. 1370–1380, Jun. 2008, PMID: 18337134, ISSN: 1093-3263.